ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ АКАДЕМИЧЕСКИЙ УНИВЕРСИТЕТ РОССИЙСКОЙ АКАДЕМИИ НАУК

На правах рукописи

Диссертация допущена к защите
Зав. кафедрой
_____
“ ”_____20... г.

Д И С С Е Р Т А Ц И Я

НА СОИСКАНИЕ УЧЕНОЙ СТЕПЕНИ

М А Г И С Т Р А

Тема: Работа с файловыми системами в операционной системе Windows с использованием драйверов операционной системы Linux

Направление: 03.04.01 – Прикладные математика и физика

Выполнил студент *Новокрещенов К.С.*

(подпись)

Руководитель:
магистр прикладной математики и физики *Баталов Е.А.*

(подпись)

Рецензент:
специалист *Борзенков П.А.*

(подпись)

Санкт-Петербург
2015 г.

Реферат

Выпускная квалификационная работа по теме «Работа с файловыми системами в операционной системе Windows с использованием драйверов операционной системы Linux» содержит 52 страницы, 4 рисунка, 4 таблицы, 13 использованных источников.

ФАЙЛОВАЯ СИСТЕМА, ДРАЙВЕР, ОПЕРАЦИОННАЯ СИСТЕМА, WINDOWS, LINUX, ВИРТУАЛЬНАЯ МАШИНА, QEMU, РАЗДЕЛЯЕМАЯ ПАМЯТЬ, ПРОТОКОЛ СЕРИАЛИЗАЦИИ ДАННЫХ, XDR, PROTOCOL BUFFERS, СОКЕТ, PCI-УСТРОЙСТВО, СЕТЕВОЕ ВЗАИМОДЕЙСТВИЕ, LIBGUESTFS, CYGWIN, MINGW

В настоящей работе рассматривается проблема организации доступа приложений операционной системы Windows к нативным файловым системам операционной системы Linux. Цель настоящей работы заключается в предоставлении приложениям операционной системы Windows возможности работать с нативными файловыми системами операционной системы Linux, поддержка которых в операционной системе Windows отсутствует.

Проведен обзор существующих файловых систем, проанализированы существующие подходы к решению обозначенной в работе проблемы. В результате проведенного анализа разработана архитектура, реализующая доступ к файловым системам операционной системы Linux в операционной системе Windows.

В итоге получена библиотека, предоставляющая приложениям операционной системы Windows определенный программный интерфейс для доступа к нативным файловым системам операционной системы Linux, размещенным как на физических устройствах хранения данных, так и на образе диска

Проведено тестирование работоспособности и производительности полученной библиотеки, измерена скорость доступа к содержимому файловой системы. Предложен и реализован ряд оптимизаций по увеличению наблюдаемой скорости доступа.

Оглавление

1 Введение

1.1 Файловые системы. Определение, функции, виды

Файловая система является ключевым компонентом операционной системы и зачастую играет решающее значение при выборе той или иной операционной системы. Файловую систему можно рассматривать как способ организации, хранения и именования данных на носителях информации в компьютере или любом другом электронном оборудовании. Без наличия файловой системы, вся информация размещалась бы на устройстве хранения данных произвольным образом, в виде одного непрерывного блока, без возможности определения, в каком месте заканчивается один блок информации и начинается другой. Файловая система определяет формат содержимого и способ физического хранения информации, которую принято группировать в виде файлов (логически неделимой совокупности данных). Таким образом, файловая система осуществляет контроль за тем, где и как файлы сохраняются на физическом устройстве и по каким правилам извлекаются обратно для использования в работе приложений.

Файловая система не только реализует определенный набор логических правил, по которым файлы размещаются на устройстве, но также хранит дополнительную информацию о каждом файле, используемую для его однозначной идентификации (имя файла) и получения сведений о некоторых его свойствах (размер, время создания, время последней модификации и т.д.). Файловые системы управляют доступом к файлам и их метаданным.

Файловые системы могут использоваться на различных устройств. Каждое устройство хранения данных использует определенный тип физического носителя. Среди них можно выделить носители с произвольным доступом такие как жесткий диск, носители с последовательным доступом (магнитные ленты), оптические носители, флэш-память и другие. Наиболее распространенными на сегодняшний день являются жесткие диски. Файловая система связывает носитель информации с одной стороны и программный интерфейс для доступа к файлам - с другой [1].

Файловая система не обязательно напрямую связана с физическим носителем информации. Существуют виртуальные файловые системы (например, procfs), файлы и каталоги внутри которых генерируются при запросе, а также сетевые файловые системы (например, Network File System (NFS)), которые являются лишь способом доступа к файлам, находящимся на удалённом компьютере.

Некоторые файловые системы были разработаны специально для решения определенного класса задач и использовании в работе конкретных приложений. Так, например, файловая система ISO 9660 была спроектирована исключительно для оптических дисков, а файловая система VMFS представляет собой кластерную файловую систему, предназначенную для хранения других файловых систем.

На сегодняшний день IT-индустрия «перегружена» огромным количеством разнообразных файловых систем. Каждая файловая система имеет свою определенную логическую структуру, характеризуется скоростью работы, надежностью и способностью к восстановлению после возникновения критических ошибок, сбоев и отказов оборудования, предоставляемой безопасностью одновременного доступа к файлам, занимаемым размером на физическом устройстве, возможностью гибкой настройки в зависимости от используемого физического носителя данных и т.д.

Каждая файловая система имеет свои преимущества и недостатки по сравнению с другими файловыми системами. Выбор той или иной файловой системе зависит от тех задач, в решении которых она будет использоваться. Например, в случае почтового сервера хорошо подойдет файловая система, спроектированная специально для работы с файлами малого размера, а при использовании в маломощных компьютерных установках отличным выбором будет использование файловой системы, потребляющей малое количество процессорного времени и не оказывающей большой нагрузки на процессор.

1.2 Нативные файловые системы

Для каждой операционной системы существует набор файловых систем, поддержка которых изначально включена в ядро операционной системы. Такие файловые системы являются нативными для данной операционной системы. Как правило, изначально каждая новая файловая система разрабатывается специально для использования в конкретной операционной системы. Наиболее известными примерами таких файловых систем являются файловые системы семейства Ext4 для операционной системы Linux, файловая система NTFS операционной системы Windows, файловая система HFS операционной системы Mac OS и многие другие.

Количество нативно поддерживаемых файловых систем сильно варьируется в зависимости от рассматриваемой операционной системы. Так, например, основное ядро операционной системы Linux поддерживает широкий спектр разнообразных файловых систем, в то время как в операционной системе Windows включена поддержка сравнительно небольшого количества файловых систем. Рассмотрим более подробно, какие файловые системы поддерживаются операционными системами Windows и Linux.

1.2.1 Файловые системы операционной системы Linux

За всё время существования и эволюции операционной системы Linux в её основное ядро была добавлена поддержка огромного количества разнообразных файловых систем[1]. Рассмотрим наиболее популярные и известные из них[2].

[1] http://en.wikipedia.org/wiki/Category:Linux_kernel-supported_file_systems
[2] http://www.howtogeek.com/howto/33552/htg-explains-which-linux-file-system-should-you-choose

1.2.1.1 Семейство файловых систем Ext

Файловая система Ext

Первая файловая система, разработанная специально для нужд операционной системы Linux. Была создана в 1992 с целью преодолеть ограничения существовавшей в то время файловой системы Minix. На сегодняшний день файловая система Ext является устаревшей и уже не поддерживается во многих дистрибутивах Linux.

Файловая система Ext2

Нежурналируемая файловая система, представляет собой продолжение развития файловой системы Ext. Включена поддержка расширенных файловых атрибутов, увеличен максимальный поддерживаемый размер физического устройства (до 2 терабайт). Поскольку журналирование не используется, в процессе своей работы данная файловая система осуществляет существенно меньше операций записи на диск, вследствие чего хорошо подходит для использования в устройствах с флеш-памятью.

Файловая система Ext3

Дальнейшее развитие семейства Ext файловых систем привело к появлению журналируемой файловой системы Ext3. По сути является расширением файловой системы Ext2, способное к журналированию. Одной из главных целей при разработке данной файловой системы была поддержка обратной совместимости. Вследствие этого переход с файловой системы Ext2 к файловой системе Ext3 не требует форматирования. На сегодняшний день Ext3 является наиболее стабильной и поддерживаемой файловой системой в среде Linux.

Файловая система Ext4

Представляет собой своеобразную попытку создать 64-хбитную Ext3, способную поддерживать больший размер файловой системы. Также добавились такие возможности как непрерывные области дискового пространства, задержка выделения пространства, онлайн дефрагментация и прочие. Обеспечивается прямая совместимость с системой Ext3 и ограниченная обратная совместимость при недоступной способности к непрерывным областям дискового пространства.

1.2.1.2 Файловая система ReiserFS

Первая попытка создать файловую систему нового поколения для Linux. Данная файловая система, представленная в 2001 году, включает в себя журналирование, возможность динамического масштабирования и лишена многих недостатков, присутствующих в файловых системах семейства Ext. Данная файловая система показывает прекрасную производительность при работе с маленькими файлами и логами, поэтому идеально подходит для использования в почтовых серверах.

1.2.1.3 Файловая система BtrFS

Файловая система, изначально разрабатываемая компанией Oracle, основана на структурах B-деревьев и работает по принципу копирование-при-записи. Предоставляет такие возможности как снимки файловой системы, контроль за целостностью данных и метаданных, прозрачное сжатие данных, журналирование, поддержка оптимизированного режима при использовании в SSD-накопителях, дефрагментация в рабочем режиме и многое другое. Немаловажным фактом является возможность перехода с файловых систем Ext3 и Ext4 на BtrFS без необходимости форматирования. Данная файловая система лишена многих недостатков, присущим предшествующим файловым системам операционной системы Linux, и призвана стать файловой системой Linux по умолчанию.

1.2.1.4 Файловая система XFS

Файловая система, разработанная компанией Silicon Graphics в 1994 году и портированная на Linux в 2001 году. Во многом похожа на файловую систему Ext4, реализует журналирование, задержку выделения пространства (как метод борьбы с фрагментацией), онлайн дефрагментацию, обладает возможностью динамического расширения. Показывает хорошую производительность при работе с большими файлами, но в остальном не обладает существенными преимуществами по сравнению с Ext4.

1.2.1.5 Файловая система JFS

Файловая система, разработанная компанией IBM в 1990 году и позже портированная на Linux. Главными особенностями данной файловой системы является низкий уровень потребления процессорного времени и хорошая производительность при работе с файлами как большого, так и малого размера. Как и файловые системы XFS и ReiserFS, предоставляет возможность динамического масштабирования, поддерживает журналирование. Вследствие низкого уровня использования процессора, хорошо подходит для применения в работе маломощных компьютеров и серверов.

1.2.1.6 Файловая система ZFS

Файловая система, разработанная компанией Sun Microsystems, первоначально для операционной системы Solaris. Среди отличительных особенностей можно выделить отсутствие фрагментации данных как таковой, возможность по управлению снимками файловой системы, пулами хранения. ZFS показывает прекрасные результаты производительности при работе с большими дисковыми массивами. Поскольку данная файловая система является проприетарной, она не может быть включена в основное ядро операционной системы Linux. Тем не менее, в Linux-системах ZFS может использоваться с помощью фреймворка FUSE[1].

[1] http://fuse.sourceforge.net/

1.2.2 Файловые системы операционной системы Windows

По сравнению с операционной системой Linux, операционная система Windows поддерживает существенно меньшее количество файловых систем[1]. К ним относятся файловые системы FAT[2], NTFS[3], exFAT[4].

1.2.2.1 FAT

Классическая архитектура файловой системы, которая из-за своей простоты всё ещё широко используется для флеш-накопителей и карт памяти. Данное семейство файловых систем (8-bit FAT, FAT-12, FAT-16, FAT-32) поддерживается почти всеми операционными системами для персональных компьютеров, включая все версии операционной системы Windows и MS-DOS. Является в некотором смысле универсальной файловой системой, предназначенной для обмена данными между компьютерами и устройствами практически любого типа.

1.2.2.2 NTFS

NTFS заменила ранее использовавшуюся в MS-DOS и Windows файловую систему FAT. Впервые появилась в операционной системе Windows-NT, выпущенной в 1993 году. NTFS поддерживает систему метаданных и использует специализированные структуры данных для хранения информации о файлах для улучшения производительности, надёжности и эффективности использования дискового пространства. NTFS имеет встроенные возможности разграничения доступа к данным для различных пользователей и групп пользователей (списки контроля доступа – Access Control Lists, ACL), а также позволяет назначать квоты (ограничения на максимальный объём дискового пространства, занимаемый теми или иными пользователями). NTFS использует систему журналирования для повышения надёжности файловой системы. Также среди особенностей данной файловой системы можно выделить поддержку жестких ссылок, разреженных файлов, шифрования, сжатия данных, точек повторной обработки (специальный тип каталогов, используемых как точки монтирования других файловых систем, символических ссылок, ссылок удаленных файловых систем).

1.2.2.3 exFAT

Проприетарная файловая система, является преемником файловой системы FAT-32. Разработана компанией Microsoft преимущественно для мобильных носителей таких как флеш-память, SSD-диски, смарткарты. Позволяет хранить файлы и использовать разделы значительно большего размера, чем в файловых системах семейства FAT. В системе exFAT также появилась возможность управления правами доступа на файлы/каталоги, а время

[1] http://en.wikipedia.org/wiki/File_system#Microsoft_Windows
[2] https://ru.wikipedia.org/wiki/FAT
[3] https://ru.wikipedia.org/wiki/NTFS
[4] http://en.wikipedia.org/wiki/ExFAT

доступа к данным уменьшилось. Файловую систему exFAT можно считать конкурентом NTFS на системах с ограниченной вычислительной мощности и памяти. Поддерживается в Windows XP, Windows Vista, Windows 7, Windows 8.

1.3 Необходимость поддержки ненативных файловых систем

Необходимость работы с файловыми системами, не являющимися нативными для используемой операционной системы, может возникнуть в ряде случаев.

Зачастую в рамках решения той или иной задачи наиболее выгодно использовать конкретную файловую систему из-за предоставляемый ею преимуществ (высокая скорость работы, поддержка журналирования, возможность шифрования данных и т.д.) по сравнению со другими файловыми системами, даже если выбранная файловая система не является нативной в используемой операционной системе. При этом не всегда удобно отказаться от используемой операционной системы в пользу той, в которой требуемая файловая система поддерживается в качестве нативной.

Многим прикладным программам, работающим в той или иной операционной системе, не всегда достаточно поддерживаемого операционной системой набора файловых систем. Существуют целые классы программного обеспечения, которым в силу своей специфики требуется работать с широким количеством файловых систем. К ним относятся:

- системы резервного копирования и восстановления данных;
- системы защиты данных, антивирусное программное обеспечение;
- менеджеры жестких дисков и разделов;
- файловые менеджеры – программы для просмотра и управления содержимым файловой системы, реализующие возможность копирования, перемещения, удаления файлов и каталогов внутри файловой системы.

В случае если поддержка файловый системы не реализована на уровне операционной системы, задача организации доступа к ней должна решаться в рамках разрабатываемого программного обеспечения.

При этом важно помнить, что чем с большим количеством ненативных файловых систем способно работать приложение, тем оно более конкурентоспособно и привлекательно для пользователя.

2 Постановка и анализ задачи

2.1 Цель работы

На сегодняшний день, операционные системы Windows и Linux являются одними из самых распространенных. Следовательно, это становится верным и для файловых систем, которые они используют в своей работе. Важно заметить, что множество нативных файловых систем операционной системы Windows является подмножеством файловых систем операционной системы Linux. Поэтому в случае необходимости приложению, работающему в операционной системе Linux, получить доступ к содержимому файловых систем, нативных для операционной системы Windows, не нужно прилагать дополнительных усилий. Практически все файловые системы операционной системы Windows поддерживаются основным ядром операционной системы Linux.

Однако при необходимости приложению Windows получить доступ к нативным файловым системам операционной системы Linux всё становится гораздо сложнее. Практически все файловые системы, нативные для операционной системы Linux, не поддерживаются в операционной системе Windows. В этом случае проблема организации доступа к требуемой файловой системе операционной системы Linux должна решаться самим Windows-приложением.

В настоящей работе решается проблема организации доступа к нативным файловым системам операционной системы Linux внутри операционной системы Windows.

Цель настоящей работы заключается в предоставлении приложениям операционной системы Windows возможности работать с файловыми системами, нативными для операционной системы Linux, но не поддерживаемыми операционной системой Windows.

2.2 Организация доступа к ненативным файловым системам

Для организации работы с любой файловой системы необходим так называемый «драйвер» файловой системы. Драйвер файловой системы можно охарактеризовать как некоторый программный компонент, интерпретирующий структуры файловой системы и предоставляющий использующим её приложениям логический иерархический вид её содержимого. Драйвер может являться частью операционной системы (в случае нативных файловых систем) либо поставляться сторонним производителем как отдельный программный модуль.

Для возможности работы с нативной файловой системой отдельному приложению не нужно прилагать никаких усилий. Как уже упоминалось выше, операционная система изначально имеет встроенные в её ядро драйверы, организующие полный доступ к нативным файловым системам, включающий в себя возможность чтения, записи,

модификации содержащихся в файловой системе каталогов и файлов. Стоит отметить, что нативные драйверы файловых систем, как правило, пишутся, тестируются и поддерживаются самими разработчиками операционной системы, в которой данная файловая система является нативной. Вследствие этого, такие драйверы обладают высокой надежностью, производительностью, эффективно реализуют все возможности файловой системы, своевременно обновляются и активно поддерживаются разработчиками в течение всего времени существования файловой системы как нативной.

Для возможности приложению работать с файловой системой, не являющейся нативной, необходимо операционной системе предоставить драйвер этой файловой системы. Поэтому проблема поиска и реализации такого драйвера должна быть решена самими разработчиками данного приложения.

Можно выделить два принципиально отличающихся друг от друга подхода к решению данной проблемы:

1. Использование «нативного» драйвера файловой системы, то есть драйвера, реализованного для той операционной системы, в которой данная файловая система является нативной;
2. Использование «портированного» драйвера файловой системы, то есть драйвера, реализованного специально для работы в целевой операционной системе.

Поскольку в настоящей работе исследуется возможность поддержки нативных файловых систем Linux в операционной системе Windows, рассмотрим оба подхода более детально на примере операционных систем Windows и Linux. В данном случае операционная система Linux является исходной (нативной), и драйверы, используемые в ней для доступа к её нативным файловым системам, мы будем называть «нативными». При этом операционная система Windows будет выступать в качестве целевой, соответственно, реализованные для работы в ней драйверы, обеспечивающие доступ к нативным файловым системам Linux, мы будем называть «портированными». Также особое внимание будет сосредоточено на анализе уже существующих в рамках каждого подхода примеров решений.

2.2.1 Использование портированного драйвера файловой системы

В рамках данного подхода существуют несколько возможных решений. Рассмотрим каждое из них в отдельности и приведем существующие примеры каждого из решений при наличии таковых.

2.2.1.1 Использование драйвера файловой системы, реализованного «с нуля»

Отличительной особенностью подавляющего числа современных файловых систем является их высокая сложность, непростое внутреннее устройство, использование в своей работе трудных и сложных алгоритмов. Это объясняется тем, что к вновь появляющимся файловым системам предъявляются высокие требования по скорости доступа к данным, эффективному использованию ресурсов, предоставляемой безопасности и многие другие. Каждая новая файловая система должна быть производительнее своих предшественников, более эффективно решать поставленные перед ней задачи, чтобы быть конкурентоспособной и более привлекательной для конечного пользователя.

Сложность внутренней организации современных файловых системы и используемых ими алгоритмов работы оказывают прямое влияние на сложность реализации полноценных драйверов файловых систем. Разработка драйвера современной файловой системы, например, такой как BtrFS, представляет собой достаточно трудоемкий процесс, требующий больших затрат ресурсов и времени. Для реализации надежного и эффективного драйвера требуется не только глубокое изучение внутреннего устройства файловой системы и принципов её работы, но и всестороннее знание особенностей операционной системы, в которой он будет работать. Стоит также упомянуть о необходимости длительного тестирования правильной работоспособности полученного драйвера перед его использованием с целью выявления допущенных при его разработке ошибок и недоработок. Также необходимо постоянно следить за изменениями файловой системы и своевременно обновлять созданную версию драйвера.

Таким образом, одним из главных недостатков данного решения является его высокая сложность и трудоемкость, обремененная необходимостью постоянно поддерживать реализованный драйвер в актуальном состоянии при обновлениях файловой системы. В случае необходимости одновременной поддержки нескольких файловых систем затраты на реализацию данного решения возрастают пропорционально количеству поддерживаемых файловых систем. К преимуществам данного решения можно отнести потенциально высокую производительность работы полученного драйвера, полностью реализующего всю функциональность файловой системы.

2.2.1.2 Использование драйвера файловой системы, реализованного сторонними разработчиками

В данном случае в зависимости от «происхождения» драйвера возможны два варианта: использование свободно распространяемого драйвера либо использование драйвера, реализованного коммерческой организацией.

В большинстве случаев, качество и надежность свободно распространяемых драйверов оставляет желать лучшего. Как правило, разработка таких драйверов чаще всего носит чисто экспериментальный характер. Зачастую в таких драйверах реализуются лишь некоторые базовые функции файловой системы, многое остается недоделанным и

недоработанным. Также характерной особенностью таких драйверов является отсутствие тщательного тестирования и поддержки со стороны разработчиков. Ошибки, допущенные при разработке такого драйвера, в процессе его использования могут привести к самым нежелательным последствиям: от ошибки времени выполнения и нежелательной перезагрузки операционной системы до отказа и повреждения оборудования с частичной или полной потерей данных. В любом случае, задачу доработки, тестирования и своевременного обновления драйвера необходимо решать конечному пользователю данного драйвера. На данный момент для некоторых файловых систем операционной системы Linux крайне сложно найти хорошую реализацию полноценного драйвера, предназначенного для работы в операционной системе Windows, иногда такого драйвера просто не существует.

Коммерческие драйвера файловой системы, как правило, лишены недостатков, связанных со свободно распространяемыми драйверами. Главным же их недостатком, очевидно, является необходимость материальных затрат на покупку лицензии для использования драйвера и поддержки со стороны разработчиков. В случае необходимости организации доступа к большому набору файловых систем данное решение может оказаться слишком невыгодным.

2.2.1.3 Использование отдельных приложений и утилит, реализующих частичный или полный доступ к файловой системе

Существуют отдельные приложения операционной системы Windows, реализующие доступ к некоторым файловым системам, не являющимися нативными. Чаще всего такие приложения реализуют ограниченную функциональность и выступают в качестве файловых обозревателей, позволяя лишь просматривать содержимое файловой системы, осуществлять чтение и копирование файлов. Главным недостатком при использовании таких программ является невозможность работать с файловой системой из других приложений. Для полноценной работы с содержащимися в файловой системы файлами и каталогами требуется вручную скопировать интересующие файлы на поддерживаемую операционной системой файловую систему и использовать в последующей работе только их копии. В качестве преимущества данного способа можно указать тот факт, что поскольку доступ к файловой системе выполняется в режиме чтения, гарантируется, что содержащиеся данные внутри файловой системы не будет повреждены, и сама файловая система останется в неизменном состоянии.

2.2.1.4 Примеры существующих решений

Подавляющее большинство из существующих на данный момент драйверов операционной системы Windows, предназначенных для работы с файловыми системами операционной системы Linux, реализовано для доступа к файловым системам семейства Ext[1].

[1] http://www.howtogeek.com/112888/3-ways-to-access-your-linux-partitions-from-windows

Драйвер Ext2fsd[1]

Свободно распространяемый драйвер операционной системы Windows для доступа к Ext2, Ext3 и Ext4 файловым системам. Предназначен для использования в Windows XP, Windows Vista, Windows 7. Предоставляет возможность работать с перечисленными файловыми системами нативно, обеспечивая другим приложениям доступ к файловой системе через букву диска[2].

Основные особенности драйвера Ext2fsd:

- поддержка чтения и записи Ext2 и Ext3 файловых систем;
- поддержка чтения Ext4 файловой системы;
- отсутствие полной поддержки журналирования файловой системы Ext3;
- отсутствие поддержки менеджера логических томов операционной системы Linux;
- отсутствие поддержки шифрования.

Данный драйвер является наиболее распространённым среди драйверов такого типа, в большинстве случаев работает стабильно, однако иногда при записи возможны потери данных[3].

Драйвер Ext2 Installable File System[4]

Свободно распространяемый драйвер операционной системы Windows для работы с файловыми системами семейства Ext. Предназначен для использования в следующих операционных системах семейства Windows: Windows NT, Windows XP, Windows Vista. Представляет собой драйвер файловой системы Ext2, полностью поддерживающий операции чтения и записи. Работает в режима ядра, то есть на том же программном уровне, что и нативные драйвера операционной системы Windows. Драйвер позволяет назначить разделам с файловой системой Ext2 букву диска, что предоставляет возможность всем приложениям взаимодействовать и работать с ними. Поскольку файловая система Ext3 поддерживает обратную совместимость с файловой системой Ext2, данный драйвер также может использоваться для организации работы с Ext3, но без поддержки журналирования. К основными недостатками данного драйвера можно отнести:

- отсутствие поддержки журналирования в файловой системе Ext3;
- отсутствие поддержки прав доступа;
- отсутствие возможности производить дефрагментацию файловой системы, как и возможности получения информации о текущей фрагментации раздела;
- отсутствие поддержки менеджера логических дисков операционной системы Linux.

[1] http://www.ext2fsd.com/
[2] http://en.wikipedia.org/wiki/Drive_letter_assignment
[3] http://sourceforge.net/projects/ext2fsd/reviews
[4] http://www.fs-driver.org/

Драйвер Paragon ExtFS for Windows[1]

Программное обеспечение компании Paragon, предоставляющий полноценный доступ к файловым системам Ext2, Ext3 и Ext4. Ключевые особенности:

- обеспечивает быстрый и прозрачный доступ к файловым системам семейства Ext, полностью поддерживает операции чтения и записи;
- предоставляет инструменты для создания и форматирования разделов с файловым системами семейства Ext;
- поддерживает работу с LVM разделами только в режиме чтения;
- поддерживает последнюю версию операционной системы Windows 8.1;
- поскольку работает в режиме ядра, обеспечивает хорошую производительность и скорость работы.

Приложение DiskInternals Linux Reader[2]

Приложение с графическим пользовательским интерфейсом операционной системы Windows для просмотра содержимого нативных файловых систем операционной системы Linux. Поддерживает чтение таких файловых систем как Ext2, Ext3, RaizerFS, Reizer4, а также HFS+ операционной системы Mac OS. Не предоставляет остальным программам возможность работать с данными файловыми системами через символ диска. Поскольку поддерживается только чтение, гарантируется, что при использовании данной программы файловая система останется в неизменном состоянии и не будет повреждена.

Приложение Ext2Read[3]

Приложение операционной системы Windows с графическим интерфейсом похожим на интерфейс файлового менеджера Windows Explorer. Используется для просмотра и чтения содержимого Ext2, Ext3 и Ext4 файловых систем. Также поддерживает возможность чтения разделов под управлением менеджера логических томов Linux.

Приложение Explore2fs[4]

Графическое приложения для просмотра содержимого Ext2 и Ext3 файловых систем. Поддерживает только возможность чтения и копирования файлов.

Утилита LTOOLS[5]

Утилита командной строки, предназначенная для чтения и записи Ext2 и Ext3 файловых систем. Также поддерживает возможность работы с файловой системой ReiserFS.

[1] https://www.paragon-software.com/home/extfs-windows-pro
[2] http://www.diskinternals.com/linux-reader/
[3] http://sourceforge.net/projects/ext2read/
[4] http://sourceforge.net/projects/explore2fspe/
[5] http://www.it.hs-esslingen.de/~zimmerma/software/ltools/ltools.html

Проект ZFS-WIN[1]

Свободно распространяемый драйвер операционной системы Windows, предназначенный для доступа к файловой системе ZFS. Позволяет монтировать раздел с файловой системой ZFS только в режиме чтения и осуществлять доступ к нему из других приложений через присвоенную разделу букву диска.

Проект WinBtrfs[2]

Драйвер операционной системы Windows, работающий в пользовательском режиме, предназначен для доступа к файловой системе BtrFS. Позволяет приложениям операционной системы Windows работать с файловой системой BtrFS только в режиме чтения.

2.2.2 Использование нативных драйверов

Нативные драйверы файловой системы лишены многих недостатков, присущих их портированным на другие операционные системы аналогам. Такие драйверы полностью поддерживают все функции и особенности файловой системы, показывают более высокие результаты производительности, поскольку изначально проектируются для использования в конкретной операционной системе. Они более надежны, качественны и активно поддерживаются разработчиками как файловой, так и операционной системы, в которой эта файловая система является нативной. Используя нативные драйверы файловой системы, практически полностью пропадает необходимость дополнительного их тестирования и отслеживания изменений файловой системы. В случае с драйверами нативных файловых систем операционной системы Linux, прежде чем включить поддержку файловой системы в основное ядро Linux, драйвер этой файловой системы проходит тщательное тестирование огромным Linux-сообществом, в состав которого входят ведущие компании IT-индустрии.

Однако поскольку драйвер относится к типу программного обеспечения, сильно зависящего от операционной системы, его нельзя просто взять и начать использовать в другой, ненативной для него, операционной системе. Для реализации этой возможности необходимо в целевой операционной системе создать окружение, эмулирующее окружение нативной для данного драйвера операционной системы. Соответственно, для использования нативных драйверов файловых систем операционной системы Linux внутри операционной системы Windows требуется создать некоторое окружение ядра Linux, которое ожидает увидеть драйвер.

Таким образом, с одной стороны, проблема поиска надежного и эффективного драйвера оказывается полностью решенной, но, с другой стороны, возникает не менее тривиальная задача – эмуляции окружения одной операционной системы внутри другой.

[1] https://code.google.com/p/zfs-win
[2] https://github.com/jgottula/WinBtrfs

Данную проблему можно решить следующими способами:

1. Реализовать программный интерфейс (API) ядра нативной для драйвера операционной системы (в нашем случае Linux) в пространстве пользователя целевой операционной системы (в нашем случае Windows);
2. Использовать виртуальную машину.

Очевидно, что сложность реализации первого способа колоссальна, особенно в случае необходимости эмуляции такой операционной системы как Linux. Разработка современной операционной системы представляет собой одну из самых сложных задач в IT-индустрии, а потому и портирование операционной системы для запуска в качестве отдельного приложения внутри другой во многом превосходит сложность портирования драйвера файловой системы.

Эмуляция окружения ядра Linux для работы нативного драйвера файловой системы операционной системы Linux в пользовательском пространстве Windows предполагает реализации как минимум:

- блочного уровня ядра Linux;
- фреймворка файловых систем;
- примитивов синхронизации;
- фреймворка файловых систем в пользовательском пространстве (FUSE).

Также стоит отметить, что вследствие того, что API ядра Linux постоянно меняется, необходимо прилагать дополнительные усилия на поддержание эмулированного ядра в актуальном состоянии.

Использование виртуальной машины для эмуляции окружения одной операционной системы внутри другой позволит использовать нативную для драйвера операционную систему без изменений. В данном случае полностью исчезает проблема портирования как драйвера, так и нативной для него операционной системы. Главным недостатком такого подхода является тот факт, что использование виртуальной машины предполагает дополнительные издержки ресурсов системы (памяти и процессорного времени).

Несмотря на то, что при реализации первого подхода мы потенциально может получить менее ресурсоемкое и более производительное решение, использование виртуальной машины выглядит наиболее привлекательным с точки зрения потраченных усилий и получаемых выгод.

Именно такой подход был выбран для реализации возможности работы с нативными файловыми системами операционной системы Linux в операционной системе Windows.

2.3 Выбор решения

Для решения задачи организации доступа к нативным файловым системам операционной системы Linux в операционной системе Windows был выбран подход, заключающийся в использовании нативных драйверов Linux с эмуляцией окружения ядра Linux посредством использования виртуальной машины. Кратко перечислим основные преимущества данного подхода по сравнению с остальными (рассмотренными ранее):

1. Отсутствие затрат на разработку драйверов для каждой из файловой систем, доступ к которым необходимо организовать;
2. Сложность реализации данного подхода никак не зависит от количества файловых систем, возможность работы с которыми нужно предоставить в операционной системе Windows;
3. Нативные драйвера операционной системы Linux реализуют полный доступ к файловой системе, поддерживают все её функции. Обладают высокой производительностью и надёжностью, активно развиваются и поддерживаются разработчиками;
4. Драйверы операционной системы Linux, как и сама операционная система Linux, относятся к свободно распространяемому программному обеспечению, следовательно, отсутствуют материальные затраты, связанные с их использованием;
5. Полное переиспользование исходного кода операционной системы Linux, отсутствие затрат на портирование.

Одним из главных недостатков данного подхода является существенные потери в производительности, а именно скорости выполнения операций чтения и записи. Также происходит высокое потребление системных ресурсов, связанных с тем, что осуществляется запуск целой виртуальной машины, внутри которой работает полноценная операционная система Linux. Существует несколько способов устранения данных недостатков, о них будет рассказано далее более подробно.

Реализация выбранного подхода будет выполнена в виде динамической библиотеки операционной системы Windows, написанной на языке программирования C. Выбор языка C в качестве языка разработки является наиболее целесообразным, поскольку:

- язык программирования C является одним из наиболее распространенных;
- код, написанный на языке C, потенциально обладает более высокой производительностью в связи с отсутствием дополнительных накладных расходов связанных с использованием интерпретатора, сборщика мусора и т.д., присутствующих в других языках программирования высокого уровня;
- многие языки программирования высокого уровня предоставляют возможность использовать библиотеки, написанные на C, что расширяет сферу применения реализуемой библиотеки.

Разрабатываемая библиотека должна предоставлять приложениям операционной системы Windows определенный программный интерфейс для возможности осуществления таких базовых операций как:

1. Чтение нативных файловых систем операционной системы Linux, расположенных как на реальном физическом устройстве, так и внутри образа диска. Под «чтением» файловой системы подразумевается просмотр содержимого каталогов, чтение файлов и копирование файлов;
2. Запись нативных файловых систем операционной системы Linux, расположенных как на реальном физическом устройстве, так и внутри образа диска. Под «записью» файловой системы подразумевается изменение содержимого каталогов, редактирование файлов, создание, перемещение, удаление файлов и каталогов внутри файловой системы.

2.4 Архитектура разрабатываемого решения

Модель работы разрабатываемой библиотеки представляет собой клиент-серверное взаимодействие и представлена на рисунке 1.

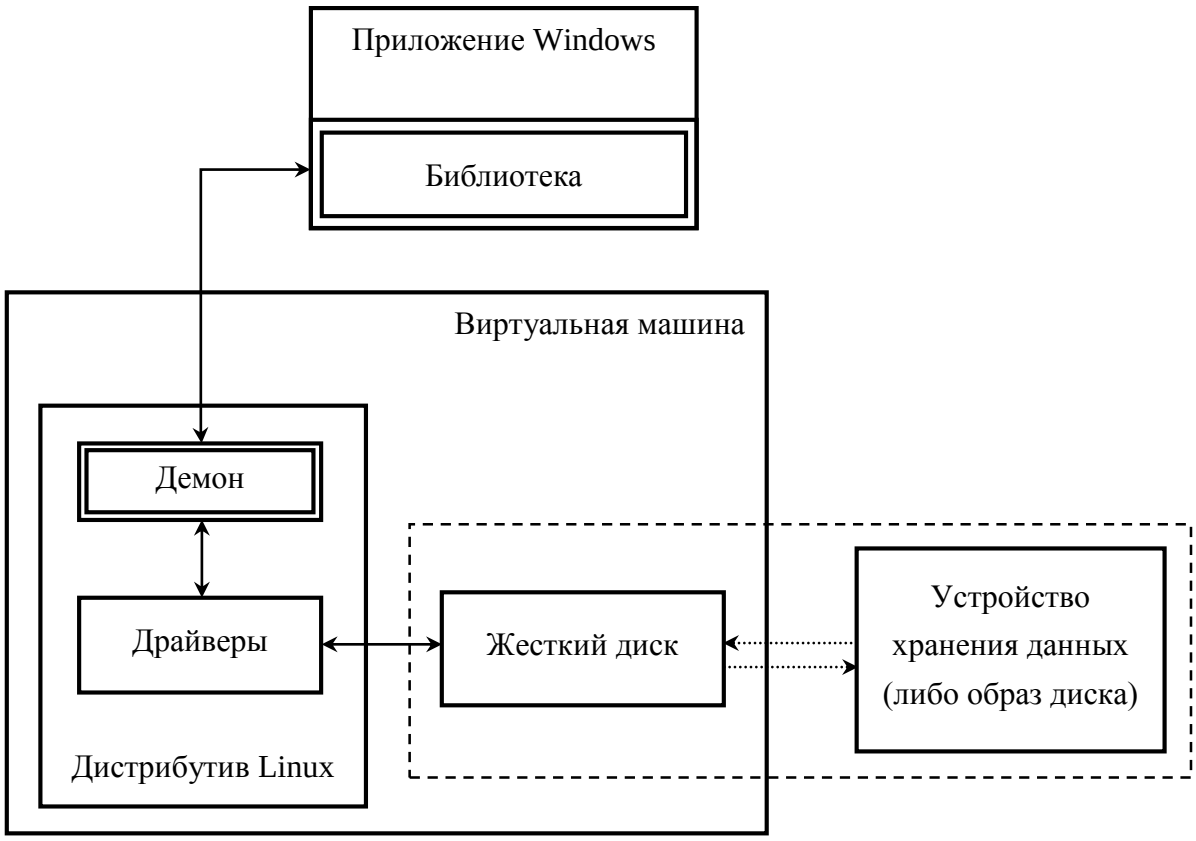


Рисунок 1 - Схема работы библиотеки

Библиотека, работающая на стороне приложения, в отдельном (дочернем) процессе запускает виртуальную машину, внутри которой загружается дистрибутив Linux. В состав загружаемого дистрибутива входят нативные драйверы файловых систем и определенный набор вспомогательных утилит для работы с ними. После загрузки операционной системы Linux в ней запускается специальный контролирующий демон, способный выполнять команды, полученные от библиотеки, по взаимодействию с требуемыми файловыми системами. Таким образом, библиотека, предоставляющая приложению определенный программный интерфейс по работе с файловой системой, выступает в роли клиента, а запущенная ею виртуальная машина с загруженной внутри операционной системой Linux и демоном выступает в роли сервера.

При запуске виртуальной машины к ней присоединяется физическое устройство (либо образ диска) с файловой системой, доступ к которой необходимо организовать. Внутри виртуальной машины присоединенное устройство выглядит как некоторое устройство хранения данных, например, как жесткий диск, с нативной для операционной системы Linux файловой системой. Стоит отметить, что в случае, если вместо реального физического устройства к виртуальной машине присоединяется образ диска, то виртуальная машины сама транслирует доступ ядра Linux к «физическому» устройству в доступ к образу диску.

Благодаря наличию в операционной системе Linux необходимых драйверов, работающий внутри виртуальной машины демон монтирует это устройство, после чего способен взаимодействовать с монтированной файловой системой и осуществлять требуемые операции чтения и записи в соответствии с командами, полученными от библиотеки, работающей на стороне приложения.

Библиотека, выступающая в роли клиента, «общается» с демоном с помощью механизма удаленного вызова процедур (RPC – Remote Procedure Call). Библиотека отправляет демону определенные команды для осуществления операций чтения и записи файловой системы и получает обратно от демона результаты их выполнения.

Рассмотренный механизм работы позволяет осуществить доступ к любой нативной файловой системе операционной системы Linux, поддержка которой включена в используемый в виртуальной машине дистрибутив Linux.

2.5 Проект libguestfs

Проект libguestfs – проект с открытым исходным кодом компании RedHat, активно развивается с 2009 года. Предназначен для использования в операционной системе Linux. Представляет собой набор утилит для доступа и модификации образов дисков виртуальных машин. Предоставляемые инструменты позволяют просматривать и редактировать файлы внутри гостевых систем, создавать образы для виртуальных машин, модифицировать их, ужимать, модифицировать таблицу разделов, управлять конфигурационными файлами,

переносить «железные» машины в виртуальную среду, переносить виртуальные машины с одного образа на другой, переносить виртуальные машины из образа на железо. Для выполнения своих операций libguestfs не требует прав суперпользователя, что является несомненным преимуществом.

Предоставляемый libguestfs набор инструментов позволяет получить доступ практически к любой из существующих файловых систем: поддерживаются все нативные файловые системы операционных систем Linux, Windows, Mac OS X, BSD, в том числе менеджер логических разделов (LVM2) Linux, дисковые разделы с таблицей разделов типа MBR и GPT, «сырые» (raw) образов дисков, CD и DVD диски, образы дисков формата ISO, SD-карты и многое другое. Также поддерживается работа с образами дисков, используемых современными виртуальными машинами. К ним относятся qcow2, VirtualBox VDI, VMWare VMDK, Hyper-V VHD/VHDX. Стоит отметить, что доступ к указанным файловым системам может осуществляться как локально, так и удаленно с помощью протоколов FTP, HTTP, SSH, iSCSI и некоторых других.

Особый интерес вызывает реализованная в проекте libguestfs библиотека, которая инкапсулирует в себе некоторую базовую функциональность, используемую во всех разработанных в рамках проекта инструментах. Данная библиотека предоставляет приложениям программный интерфейс для доступа и работы с упомянутыми ранее файловыми системами и образами дисков. Библиотека реализована на языке C, имеет привязки более чем к 10 языкам таким как Ocaml, Python, Ruby, Java, Haskell и лругие.

2.5.1 Модель и принцип работы библиотеки libguestfs

Предоставляемая проектом libguestfs библиотека в своей работе использует архитектуру, аналогичную рассмотренной в 2.4. Схема работы библиотеки libguestfs представлена на рисунке 2.

В качестве виртуальной машины используется QEMU[1] и UML[2]. Также есть возможность использовать libvirt либо подключиться к уже запущенному внутри виртуальной машины демону.

Взаимодействие библиотеки с демоном осуществляется через Unix-сокеты посредством удаленного вызова процедур по протоколу XDR [2]. В большинстве случаев каждому вызову функции библиотеки соответствует отправка определенной команды демону.

[1] http://wiki.qemu.org/Main_Page
[2] http://user-mode-linux.sourceforge.net

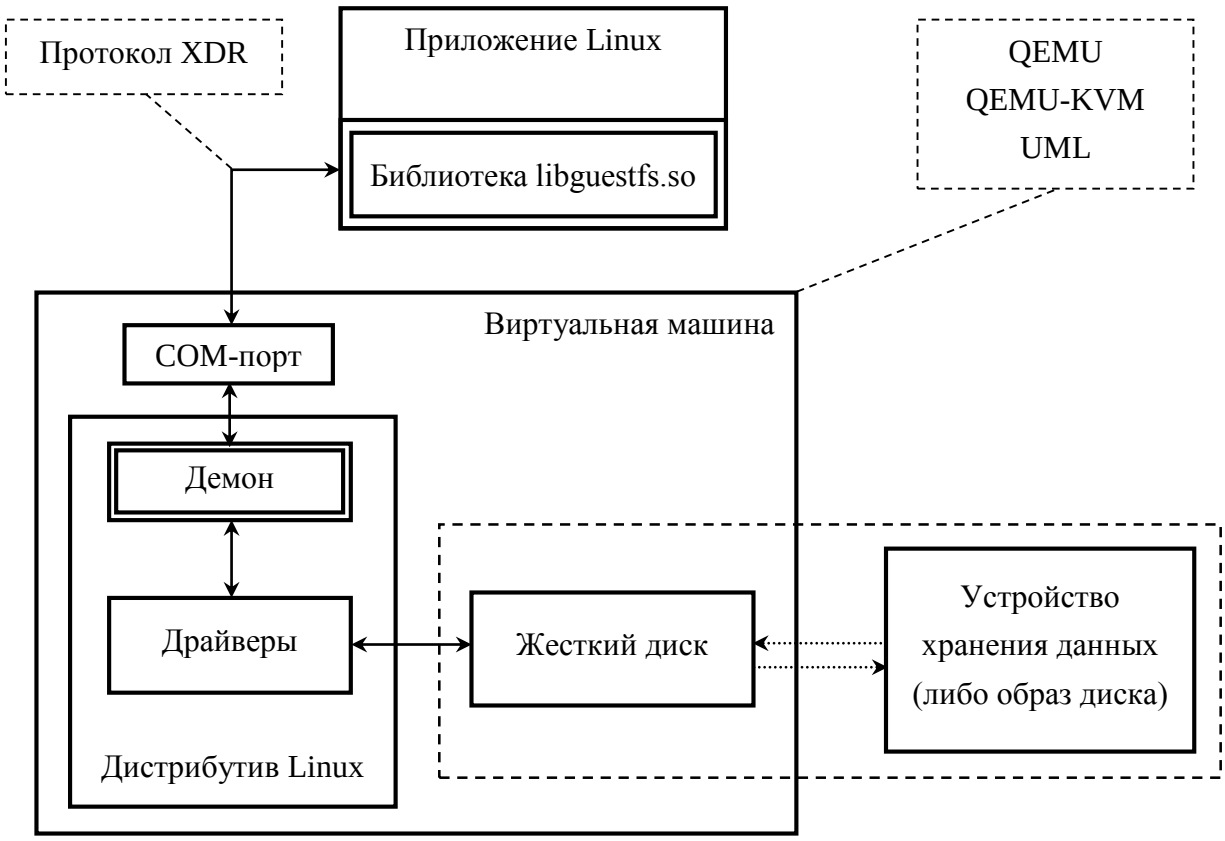


Рисунок 2 – Схема работы библиотеки libguestfs

В качестве дистрибутива, загружаемого внутри виртуальной машины, используется дистрибутив Linux, полученный с помощью вызова утилиты supermin[1]. Утилита supermin, представляющее собой приложение операционной системы Linux, «собирает» дистрибутив на основе операционной системы Linux, внутри которой запускается. Результатом её исполнения является дистрибутив Linux, в состав которого входит:

1. Ядро операционной системы Linux, представляющее собой ядро операционной системы хоста;
2. Диск в оперативной памяти для начальной инициализации (Initial RAM Disk[2], или initrd), предназначенный для загрузки необходимых модулей ядра и монтирования корневой файловой системы [3];
3. Образ диска с файловой системой типа Ext2, содержащей корневое дерево файлов операционной системы Linux, драйверы файловых систем и набор необходимых утилит для работы с файловыми системами.

В документации к libguestfs [4] вводится концепция «устройство» (appliance). «Устройство» представляет собой совокупность виртуальной машины с загруженной внутри неё операционной системой Linux, запущенного демона и монтированного корневого образа диска.

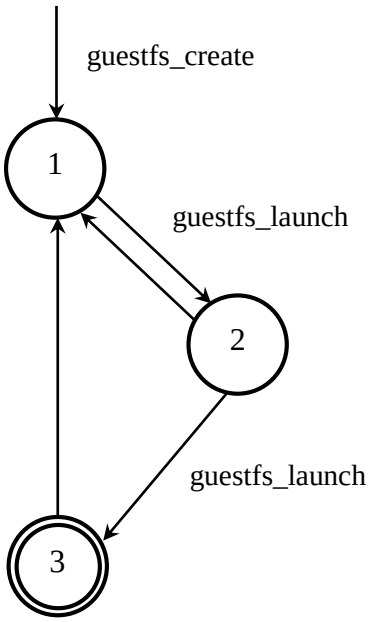
[1] http://libguestfs.org/supermin.1.html
[2] https://ru.wikipedia.org/wiki/Initrd

Подготовка библиотеки к работе заключается в инициализации «устройства». Процесс инициализации «устройства» включает:

- создание дистрибутива Linux с помощью утилиты supermin;
- запуск виртуальной машины;
- загрузка initrd;
- монтирование корневого образа диска, содержащего операционную систему Linux;
- запуск демона.

«Устройство» может находиться в одном из трех состояний: «конфигурация», «запуск», «готовность к работе». Правильная работа с устройством предполагает следующую смену состояний: «конфигурация», «запуск», «готовность к работе»

Конечный автомат состояний «устройства» представлен на рисунке 3.



1 – конфигурация; 2 – запуск;
3 – готовность к работе

Рисунок 3 – Автомат состояний «устройства»

2.5.2 Программный интерфейс библиотеки libguestfs

Для возможности внутри пользовательского приложения осуществлять настройку параметров запуска виртуальной машины и впоследствии взаимодействовать с запущенным внутри нее демоном библиотека libguestfs предоставляет специальный обработчик – объект типа `guestfs_h`, который создается при помощи вызова функции `guestfs_create`. Данный обработчик характеризует текущее состояние библиотеки и создаваемого ею «устройства» и используется при вызове всех функций, предоставляемый библиотекой libguestfs в качестве программного интерфейса.

Программный интерфейс библиотеки предлагает один способ перейти из состояния «конфигурация» через состояние «запуск» в состояние «готов к работе» - посредством вызова функции `guestfs_launch` (см. рисунок 3). Вызов функции `guestfs_launch` блокирует выполнение приложения до тех пор, пока не закончится процесс инициализации устройства и оно не перейдет в состояние «готовность к работе». `guestfs_launch` в процессе своего вызова изменяет состояние устройства из «конфигурация» в «запуск», пока происходит запуск виртуальной машины, загрузка операционной системы Linux и запуск демона.

После того, как «устройство» перешло в состояние «готовность к работе», можно осуществлять доступ к требуемой файловой системе. Для этого необходимо монтировать устройство, на котором она расположена, с помощью функции `guestfs_mount`, после чего с помощью вызова соответствующих функций выполнять операции по чтению и записи файловой системы.

2.6 Задачи работы

Поскольку в рамках выбранного подхода организации доступа к нативным файловым системам операционной системы Linux в операционной системе Windows архитектура и принцип работы библиотеки libguestfs соответствует разрабатываемой, было принято решение портировать исходный код библиотеки libguestfs для возможности её работы в операционной системе Windows.

Таким образом, выполняемые в рамках настоящей работы задачи были сформулированы следующим образом:

1. Выбрать подходящую виртуальную машину для запуска операционной системы Linux внутри операционной системы Windows;
2. Портировать библиотеку libguestfs для работы в операционной системе Windows;
3. Предложить и реализовать ряд оптимизаций по повышению производительности и скорости работы библиотеки;
4. Сравнить производительность работы портированной библиотеки с исходной.

3 Выбор виртуальной машины

Основными кандидатами на использование в качестве виртуальной машины являлись VirtualBox[1], coLinux[2] и QEMU. Рассмотрим особенности каждой из них более подробно.

3.1 Виртуальная машина VirtualBox

Программный продукт виртуализации для операционных систем Microsoft Windows, Linux, FreeBSD, Mac OS X, Solaris/OpenSolaris, ReactOS, DOS и других. Разрабатывается компанией Oracle. Ключевые возможности:

- кроссплатформенность;
- поддерживает аппаратную виртуализацию;
- высокая производительность работы гостевой операционной системы.

3.2 Виртуальная машина coLinux

Технология, позволяющая нативно запускать операционную систему Linux на операционной системе Windows 2000/XP/Vista/7. coLinux использует модифицированный ядро Linux для запуска совместно с другой операционной системой на одной и той же машине. Такая возможность достигается благодаря использованию специального 32-битный драйвер Windows для отображения системных вызовов Linux в вызовы Windows.

Отличительные особенности:

- позволяет добиться практически той же функциональности и производительности как при работе обычной операционной системы Linux, запущенной на той же самой машине в одиночку;
- требует прав суперпользователя, работает в привилегированном режиме процессора, в случае ошибок возможен крах всей системы;
- не поддерживает работу в 64-битных операционных системах.

3.3 Виртуальная машина QEMU

Свободно распространяемая виртуальная машина с открытым исходным кодом для эмуляции аппаратного обеспечения различных платформ. Включает в себя эмуляцию процессоров Intel x86 и устройств ввода-вывода. Используется как чистый эмулятор и как нативная виртуальная машина (в x86 и x86-64 архитектурах). Работает в операционных системах Linux, Windows, Syllable, FreeBSD, FreeDOS, Mac OS X, QNX, Android и других.

[1] https://www.virtualbox.org/
[2] http://www.colinux.org/

В качестве виртуальной машины было принято решение использовать QEMU, поскольку данная виртуальная машина обладает следующими преимуществами:

- свободно распространяется, имеется возможность сборки и использования в операционной системе Windows;
- работает в пространстве пользователя, не требует прав администратора;
- изначально поддерживается в библиотеке libguestfs.

4 Портирование библиотеки libguestfs в Windows

4.1 Выбор среды разработки

Для портирования библиотеки необходимо выбрать подходящую среду разработки.

Изначально планировалось использовать «родную» для операционной системы Windows среду разработки Visual C++, разрабатываемую компанией Microsoft и предназначенную для разработки нативных Windows-приложений на языке C и C++.

Однако использование Visual C++ оказалось невыгодным вследствие следующих причин:

1. Отсутствие поддержки расширений языка C как в GCC

   Поскольку проект libguestfs изначально разрабатывался для работы в Linux, сборка библиотеки libguestfs подразумевает использование инструментов GCC. Предлагаемые ими расширения языка C, такие как, например, атрибуты функций[1], активно применяется в исходной коде библиотеки.

2. Отсутствие поддержки системы сборки libguestfs

   В проекте libguestfs применяется система сборки, основанная на использовании инструментов GNU Autotools. В случае использования среды Visual C++ возникает необходимость в разработке собственной системы сборки при невозможности интегрироваться в уже существующую систему сборки проекта libguestfs.

Отказ от использования Visual C++ предполагает необходимость использования таких инструментов разработки, которые лишены рассмотренных выше недостатков Visual C++, а именно: предоставляют достойный Windows-аналог компилятора GCC и, желательно, поддерживают систему сборки GNU. С точки зрения выдвинутых требований наиболее подходящими средами разработки оказались Cygwin[2] и MinGW[3].

[1] https://gcc.gnu.org/onlinedocs/gcc/C-Extensions.html#C-Extensions
[2] https://www.cygwin.com
[3] http://www.mingw.org

4.2 Портирование libguestfs с использованием Cygwin

Cygwin – проект компании RedHat, представляет собой Unix-подобную среду и интерфейс командной строки для Microsoft Windows. Cygwin обеспечивает тесную интеграцию Windows приложений, данных и ресурсов с приложениями, данными и ресурсами Unix-подобной среды. Ключевой частью Cygwin является его динамически подключаемая библиотека cygwin1.dll, которая обеспечивает совместимость интерфейса прикладного программирования (API) и реализует значительную часть стандарта POSIX на основе системных вызовов Win32. Кроме того, Cygwin включает в себя инструменты разработки GNU для выполнения основных задач программирования, а также и некоторые прикладные программы, эквивалентные базовым программам Unix. Иными словами, Cygwin представляет собой удобную среду для запуска Unix-приложений в Windows практически без изменения их исходного кода

Первоначально, портирование библиотеки было осуществлено с использованием Cygwin, поскольку благодаря реализуемому внутри него POSIX окружению процесс портирования Unix-приложений для запуска в операционной системе Windows должен быть менее трудоемким. Это оказалось верным и для библиотеки libguestfs.

Процесс портирования библиотеки libguestfs для запуска в Cygwin состоял из двух этапов:

1. Сборка библиотеки libguestfs в Cygwin;
2. Изменение исходного кода библиотеки libguestfs.

4.2.1 Сборка библиотеки libguestfs в Cygwin

В рамках данного портирования были осуществлена сборка библиотеки libguestfs в Cygwin, которая включала следующие этапы:

1. Изучение системы сборки проекта libguestfs;
2. Внесение изменений в систему сборки проекта libguestfs

   Поскольку главной целью является сборка библиотеки libguestfs, генерация и сборка многих утилит, реализованных в рамках данного проекта, была отключена. Также была отключена проверка некоторых зависимостей, которые не являются критичными для сборки библиотеки.

3. Разрешение внешних зависимостей

   Проект libguestfs имеет большое количество зависимостей от сторонних библиотек. Многие из них были разрешены с помощью встроенного в установщик Cygwin менеджера пакетов. В случае отсутствия библиотеки (либо утилиты) в менеджере пакетов поиск и сборка библиотеки осуществлялась «вручную». Среди таких зависимостей оказались утилита supermin, менеджер пакетов ocamlfind и другие

4.2.2 Изменение исходного кода библиотеки libguestfs

В исходный код библиотеки libguestfs также потребовалось внести некоторое количество изменений. Была изменена процедура запуска виртуальной машины: в связи с отсутствием в используемом Windows-дистрибутиве QEMU поддержки Unix-сокетов, они были заменены на TCP-сокеты. Также был отключен вызов утилиты supermin, который в исходной реализации библиотеки происходил с целью сгенерировать дистрибутив Linux для его использования в запускаемой виртуальной машине. Вместо этого происходил поиск заранее подготовленного дистрибутива Linux по пути, указанному в директиве `GUESTFS_DEFAULT_PATH`.

4.2.3 Результаты портирования

В результате данного портирования была реализована возможность использования библиотеки libguestfs приложениями операционной системы Windows, работающими в среде Cygwin.

Наряду с полученным положительным результатом, существенным недостатком данного портирования оказался тот факт, что после установки всех необходимых библиотек и утилит размер дистрибутива Cygwin оказался слишком большим – порядка 1 ГБ. С целью устранить зависимость портированной библиотеки от Cygwin-окружения было принято решение осуществить нативное портирование библиотеки libguestfs с использованием инструментария, предлагаемого в рамках проекта MinGW.

4.3 Нативное портирование библиотеки libguestfs с использованием MinGW

Проект MinGW[1] представляет собой программную среду для разработки приложений операционной системы Windows. В рамках проекта предоставляется нативный программный порт компилятора GCC и утилит GNU Binutils под Windows вместе с набором свободно распространяемых библиотек импорта и заголовочных файлов для Windows API. Также в MinGW включены расширения для библиотеки времени выполнения Visual C++ для поддержки функциональности C99. Предоставляемые в рамках данного проекта инструменты позволяют разработчикам создавать нативные приложения Windows, которые дополнительно не зависят от сторонних библиотек[2].

[1] https://ru.wikipedia.org/wiki/MinGW
[2] http://www.mingw.org

4.3.1 Портирование исходного кода библиотеки libguestfs

Поскольку одной их главных задач являлось сохранение возможности сборки и использования портированной версии библиотеки libguestfs в операционной системе Linux, портирование библиотеки для работы в операционной системе Windows предполагало локализацию платформозависимого кода и сокрытие его внутри кроссплатформенных функций и интерфейсов. Ниже приведены наиболее существенные изменения, внесенные в исходный код библиотеки libguestfs в ходе нативного портирования на Windows.

4.3.1.1 Реализация кроссплатформенных интерфейсов

После изучения исходного кода библиотеки libguestfs для удобства портирования было принято решение реализовать библиотеку, содержащую в себе кроссплатформенные определения некоторых базовых концепций операционных систем. В состав библиотеки вошли такие абстракции как:

- процесс (`os_process_info_t`);
- сокет (`os_socket_t`, `os_socket_info`);
- однонаправленный канал (`os_pipe_t`, `os_pipe_end_t`);
- двунаправленный канал (`os_socketpair_t`, `os_socketpair_end_t`).

Для каждой из приведенных структур были также реализованы вспомогательные функции, упрощающие работу с ними. Добавленные функции реализуют как базовые операции, такие как инициализация, проверка на валидность, присвоение значения по умолчанию, так и некоторые уникальные для структуры операции (например, `os_process_info__kill` для `os_process_info_t`, `os_pipe_end__close` для `os_pipe_end_t` и т.д.).

4.3.1.2 Реализация отдельных POSIX функций для работы в Windows

Используемые в реализации библиотеки функции, предоставляемые в рамках стандарта POSIX или относящиеся к расширениям GNU, аналогов которым нет в операционной системе Windows, были реализованы на основе приведенной к ним документации [5] и оформлены в виде отдельной библиотеки операционной системы Windows.

В состав полученной библиотеки вошли следующие функции:

- `asprintf`, `vasprintf`;
- `symlink`, `strndup`;
- `realpath`;
- `memmem`, `open_memstream`;
- `getusername`, `getusid`, `getuid`, `geteuid`;
- `strerror_r`.

4.3.1.3 Портирование модуля `command.c`

Для исполнения системных команд операционной системы в библиотеке libguestfs используется абстракция command. Объект типа `struct command` используется для выполнения внешних системных программ и пользовательских приложений, и получения результатов их исполнения. Объект типа `struct command` позволяет также получать данные стандартного потока вывода и стандартного потока вывода ошибок, генерируемых во время выполнения внешней команды.

Особое внимание стоит обратить на то, как происходит вызов внешней команды. В модуле command.c для запуска внешней команды, представленной структурой `struct command`, используется функция `run_command`. Исходной реализация данной функции основана на вызове функции `fork`, внутри которой происходит вызов функции `execv` (либо `system`), которой в качестве входного параметра передается строка команды, полученная от процесса-родителя. Для получения данных со стандартных потоков используются однонаправленные POSIX каналы, создаваемые с помощью вызова функции `pipe`. Общая структура запуска внешней команды внутри исходной функции `run_command` имеет следующий вид:

```
cmd->pid = fork ();
...
if (cmd->pid > 0) {
...
}

/* Child process. */
...
/* Run the command. */
switch (cmd->style) {
case COMMAND_STYLE_EXECV:
  execvp (cmd->argv.argv[0], cmd->argv.argv);
...

case COMMAND_STYLE_SYSTEM:
  r = system (cmd->string.str);
...

case COMMAND_STYLE_NOT_SELECTED:
  abort ();
}
/*NOTREACHED*/
```

Используемый для вызова внешней команды `fork` с последующим вызовом функции `execv` (либо `system`) был заменен на подходящую по семантике исполнения функцию `CreateProcess`, предлагаемую в рамках Windows API. Для получения данных со стандартных потоков исполнения использовались анонимные каналы операционной системы Windows, один из концов которых назначался в качестве соответствующего стандартного потока (либо ввода, либо вывода, либо вывода ошибок) дочернего процесса,

создаваемого функцией `CreateProcess`. Общая структура запуска внешней команды внутри портированной для работы в Windows функции `run_command` имеет следующий вид:

```
switch (cmd->style) {
case COMMAND_STYLE_EXECV:
  cmdline = guestfs___join_strings (" ", cmd->argv.argv);
  break;

case COMMAND_STYLE_SYSTEM:
  cmdline = strdup (cmd->string.str);
  break;

case COMMAND_STYLE_NOT_SELECTED:
  abort();
}
/*NOTREACHED*/

bSuccess = CreateProcess (NULL,
    cmdline,
    NULL,
    NULL,
    TRUE,
    0,
    NULL,
    NULL,
    &siStartInfo,
    &cmd->pid);
```

Основные отличия в исходной и кроссплатформенной реализациях структуры `struct command` представлены ниже в таблице 1.

Таблица 1 – Структура struct command

Исходная реализация	Кроссплатформенная реализация

`struct command`
`{`
`  guestfs_h *g;`
`  ...`
 	
`  /* Capture errors to the error log`
`   * (defaults to true). */`
`  bool capture_errors;`
`  int errorfd;`
`  ...`
 	
`  void *stdout_data;`
`  int outfd;`
`  ...`
 	
`  /* PID of subprocess (if > 0). */`
`  pid_t pid;`
`};`
	`struct command`
	`{`
	`  guestfs_h *g;`
	`  ...`
	
	`  /* Capture errors to the error log`
	`   * (defaults to true). */`
	`  bool capture_errors;`
	`  os_pipe_end_t errorfd;`
	`  ...`
	
	`  void *stdout_data;`
	`  os_pipe_end_t outfd;`
	`  ...`
	
	`  /* PID of subprocess (if > 0). */`
	`  os_process_info_t pid;`
	`};`

В соответствии с представленными изменениями для остальных функций модуля command.c. также были созданы соответствующие аналоги для работы в операционной системе Windows.

4.3.1.4 Портирование модуля conn-socket.c

Для описания сетевого соединения библиотеки с демоном, работающего внутри виртуальной машины, используется структура `struct connection_socket`, расположенная в модуле conn-socket.c вместе с остальными функциями, использующие её в своей работе.

Исходная реализация структуры `struct connection_socket` приведена ниже.

```
struct connection_socket {
    const struct connection_ops *ops;

    /* Appliance console (for debug info). */
    int console_sock;

    /* Daemon communications socket. */
    int daemon_sock;

   /* Socket for accepting a connection from the daemon.
    * Only used before and during accept_connection.
    */
    int daemon_accept_sock;
};
```

Поле `daemon_accept_sock` описывает прослушивающий сокет для установления соединения с демоном, и `daemon_sock` используется для взаимодействия с демоном после установки соединения. Поле `console_sock` предназначено для получения данных с потоков стандартного ввода, стандартного вывода и стандартного вывода ошибок и представляет собой один их концов двунаправленного канала, создаваемого с помощью `socketpair`.

Кроссплатформенная реализация структуры struct connection_socket приведена ниже.

```
struct connection_socket {

    const struct connection_ops *ops;

    /* Appliance console (for debug info). */
    os_socketpair_end_t console_sock;

    /* Daemon communications socket. */
    os_socket_t daemon_sock;

   /* Socket for accepting a connection from the daemon.
    * Only used before and during accept_connection.
    */
    os_socket_t daemon_accept_sock;
};
```

В случае операционной системы Windows поля `daemon_sock` и `daemon_accept_sock` представлены типом `SOCKET`. Поле `console_sock` описывается типом `HANDLE` и представляет собой один из концов анонимного канала. Для получения данных с потоков стандартного ввода, вывода и вывода ошибок используется тот же механизм, как и в случае портирования рассмотренного выше модуля command.c.

В соответствии с приведенными изменениями была создана отдельная реализация всех функций модуля conn-socket.c для работы в Windows.

4.3.1.5 Изменение процедуры получения дистрибутива Linux

В исходной реализации библиотеки libguestfs перед запуском виртуальной машины происходит вызов утилиты supermin, в результате которого создается дистрибутив Linux, используемый впоследствии внутри виртуальной машины. Дистрибутив Linux генерируется на основе Linux, установленного на хосте. Поскольку в случае операционной системы Windows вызов утилиты supermin является бессмысленным и, очевидно, приведет к ошибке, были изменена процедура получения дистрибутива Linux. Вызов утилиты supermin для построения дистрибутива Linux был отключен, вместо этого осуществляется поиск и использование заранее подготовленного дистрибутива, расположенного по пути GUESTFS_DEFAULT_PATH/appliance.

4.3.1.6 Изменение процедуры подготовки и запуска виртуальной машины

Подготовка необходимого окружения и запуск виртуальной машины описываются, главным образом, функцией `launch_direct`, расположенной внутри модуля launch-direct.c. В работе данной функции можно выделить следующие этапы:

1. Создание прослушивающего сокета;
2. Создание канала для получения данных с потоков стандартного ввода, вывода и вывода ошибок дочернего процесса, внутри которого запускается виртуальная машина;
3. Формирование и настройка параметров запуска виртуальной машины;
4. Создание дочернего процесса и запуск внутри него виртуальной машины с предварительным перенаправлением стандартных потоков (ввода, вывода, вывода ошибок) в один из концов канала, созданного на предыдущем этапе;
5. Принятие соединения от демона на прослушивающем сокете и создание сокета для взаимодействия с демоном.

Поскольку внутри каждого из этапов используются платформозависимые возможности операционной системы, которые отличаются в операционных системах Windows и Linux, реализация каждого из этапов была скрыта внутри соответствующей кроссплатформенной функции.

Особенности работы каждого из этапов и основные изменения, внесенные в их исходную реализацию для возможности работы в операционной системе Windows, приведены ниже.

Создание прослушивающего сокета

Реализация данного этапа была оформлена в виде кроссплатформенной функции `create_daemon_accept_socket`.

Как упоминалось ранее, в исходной реализации библиотеки libguestfs взаимодействие библиотеки с демоном осуществляется через Unix-сокеты. Соответственно, для принятия соединения от демона создается прослушивающий Unix-сокет.

Порт для работы в операционной системе Windows заключается в использовании Windows-сокетов типа `SOCKET`. Создание прослушивающего сокета было осуществлено с помощью соответствующих функций по работе с сокетами, предлагаемых в рамках Windows API.

Создание канала для получения данных со стандартных потоков

Реализация данного этапа была оформлена в виде кроссплатформенной функции `open_socketpair.`

В исходной реализации библиотеки libguestfs для получения данных с потоков стандартного ввода, вывода и вывода ошибок дочернего процесса, внутри которого запускается виртуальная машина, создается двунаправленный канал с помощью вызова функции `socketpair`.

В качестве двунаправленных каналов в операционной системе Windows используются именованные каналы (Named Pipes[1]), которые относятся к глобальным объектам Windows. Первоначально было принято использовать именно их. Однако после внимательного изучения семантики использования создаваемого в библиотеке двунаправленного канала было обнаружено, что со стороны библиотеки используется только операция чтения. Поэтому порт данного этапа в Windows использует более простые анонимные каналы.

Формирование и настройка параметров запуска виртуальной машины

На данном этапе работы в функции `launch_direct` происходит формирование строки команды для запуска виртуальной машины. Формирование строки команды заключается в добавлении необходимых флагов и опций, определяющих способ запуска виртуальной машины.

В частности, для создания внутри виртуальной машины сокета, используемого демоном для «общения» с библиотекой, используется опция `–chardev socket`, после которой указывается информация о прослушивающем сокете, созданном на стороне библиотеки. В связи с изменениями в типе используемого сокета формирование данной опции было оформлены в виде кроссплатформенной функции `init_guestfsd_socket_info.`

[1] https://msdn.microsoft.com/en-us/library/windows/desktop/aa365590(v=vs.85).aspx

Создание дочернего процесса и запуск виртуальной машины с перенаправлением стандартных потоков

Реализация данного этапа была оформлена в виде кроссплатформенной функции `create_process_with_redirected_output`.

На данном этапе происходит запуск дочернего процесса, внутри которого происходит запуск виртуальной машины по сформированной на предыдущем этапе команде с предварительным перенаправлением стандартных потоков исполнения в один из концов созданного ранее канала. В исходной реализации используется вызов функции `fork` с последующим вызовом функции `execv`. В портированной для Windows версии используется `CreateProcess`. Особенности реализации данного этапа во многом аналогичны функции `run-command` модуля command.c.

Принятие соединения от демона

После запуска виртуальной машины для принятия соединения от запущенного внутри неё демона в функции `launch_direct` используется вызов функции `accept_connection` модуля conn-socket.c. Кроссплатформенная реализация данной функции уже была осуществлена при портировании модуля conn-socket.c.

Таким образом, для каждой из полученных кроссплатформенных функций было создано две реализации: одна - для операционной системы Linux, другая – для операционной системы Windows. В случае работы библиотеки в операционной системе Linux при вызове данных функций исполняется исходный библиотеки. Соответственно, при работе библиотеки в операционной системе Windows исполняется портированный аналог функции.

4.3.2 Интеграция в систему сборки проекта libguestfs

В рамках проекта MinGW реализуется программный компонент Minimal SYStem (MSYS). Он предоставляет win32-порты окружения легковесной Unix-подобной оболочки, включающей набор инструментов POSIX, достаточный для использования системы сборки GNU Autotools [6].

Как упоминалось ранее, проект libguestfs использует систему сборки GNU Autotools. Данная система сборки основана на запуске специального скрипта configure, результатом исполнения которого является конфигурационный файл config.h и получаемое на основе файлов Makifile.in дерево Makefile-ов. Более подробно о принципах работы данной системы сборки и используемых инструментах рассказывается в [7-9].

В исходный процесс сборки был добавлен режим mingwport, для включения которого в используется опция `--enable-mingwport`, передаваемая в качестве одного из входных параметров скрипта configure.

```
AC_ARG_ENABLE([mingwport],
   [AS_HELP_STRING([--enable-mingwport],
                   [enable mingwport build mode])],
   [case $enableval in
    yes|no) ;;
    *)      AC_MSG_ERROR([bad value $enableval for mingwport option]) ;;
    esac
    mingwport=$enableval],
   [mingwport=no]
)

AM_CONDITIONAL([ENABLE_MINGWPORT], [test "$mingwport" = yes])
```

В режиме mingwport отключена проверка зависимостей, не являющимися необходимыми для сборки библиотеки, а также отключена сборка некоторых инструментов, реализуемых в рамках проекта. Данный режим реализован для использования как в операционной системе Windows, так и в операционной системе Linux.

```
dnl Check for libguestfs-mingw-port
if test "$mingwport" = yes; then
    AC_MSG_NOTICE([libguestfs-mingw-port enabled])
    case $host in
    *mingw*) AC_MSG_NOTICE([mingw environment has been detected])
             [windows_host=yes] ;;
    *linux*) AC_MSG_NOTICE([linux environment has been detected])
             [linux_host=yes] ;;
    *)       AC_MSG_WARN([unknown environment has been detected]) ;;
    esac
fi
```

Процесс сборки библиотеки в режиме mingwport в операционных системах Windows и Linux немного отличается друг от друга. В обеих операционных системах происходит сборка основной библиотеки libguestfs и добавленной в проект кроссплатформенной библиотеки osdep. В операционной системе Windows, помимо этого, осуществляется сборка вспомогательной библиотеки winport, в которой реализованы Windows-аналоги некоторых функций стандарта POSIX и расширений GNU. В операционной системе Linux добавлена сборка демона и построение с помощью вызова утилиты supermin дистрибутива Linux, используемого в виртуальной машине.

4.3.3 Результаты портирования

В результате данного портирования была получена кроссплатформенная реализация библиотеки libguestfs, позволяющая использовать её в своей работе приложениям операционных систем Windows и Linux. Сохранена возможность сборки полученной библиотеки средствами GNU Autotools.

5 Улучшение производительности работы библиотеки libguestfs

Один из главных недостатков реализуемого в настоящей работе подхода по организации доступа к нативным файловым системам Linux в операционной системе Windows заключается в низкой скорости выполнения операций чтения и записи, связанной с использованием виртуальной машины и передачей файлов через сокеты. Получаемая скорость чтения и записи файлов ограничена скоростью сетевого взаимодействия и далека от той, которую можно получить, взаимодействуя с требуемой файловой системой непосредственно, без промежуточного «звена» в виде виртуальной машины.

С целью повысить скорость выполнения операций чтения и записи файловой системы было принято решение реализовать следующие оптимизации:

1. Замена протокола взаимодействия с XDR на Protocol Buffers;
2. Передача файлов через общую память.

5.1 Замена протокола взаимодействия

Одной из главных целей, преследуемых при замене протокола взаимодействия, помимо потенциального увеличения скорости передачи файлов, была необходимость оценить, насколько сильно используемый протокол сериализации данных оказывает влияние на скорость передачи данных.

5.1.1 XDR

В исходной реализации libguestfs обмен данными между библиотекой и демоном осуществляется по протоколу XDR. XDR представляет собой международный стандарт передачи данных в Интернете, используемый в различных RFC[1] для описания типов. Он позволяет данным быть упакованными не зависящим от архитектуры способом, таким образом, данные могут передаваться между гетерогенными компьютерными системами.

В операционной системе Linux для работы с этим протоколом используется библиотека xdr, реализованная компанией Sun Microsystems. Данная библиотека изначально включена во многие распространяемые дистрибутивы Linux, такие Ubuntu, Debian и другие.

К сожалению, реализованная компанией Sun Microsystems версия библиотеки для использования в операционной системе Windows является проприетарной[2].

[1] https://ru.wikipedia.org/wiki/RFC
[2] http://www.onc-rpc-xdr.com/products/rpc/rpc-windows-vc-c-delphi.asp

Среди свободно распространяемых решений для работы с протоколом XDR в операционной системе Windows можно выделить следующие:

- библиотека PortableXDR, реализованная компанией RedHat;
- библиотека, реализованная в рамках проекта bsd-xdr.

Приведенные библиотеки реализуют стандарт XDR, определенный в RFC 4506 [10] в 2006 году.

Стоит отметить, что XDR изначально обладает очень хорошей производительностью и скоростью работы с точки зрения сериализации данных, в связи с чем далеко не из каждых существующих на сегодняшний день протоколов имеет смысл использовать в качестве замены XDR.

5.1.2 Protocol Buffers

Protocol Buffers[1] представляет собой язык описания данных, предоставляющий гибкий и эффективный механизм сериализации структур данных. Разработан компанией Google как альтернатива XML, но меньше, быстрее и проще. Данный протокол предназначен для использования в C++, Java и Python.

Protocol Buffers и XDR во многом похожи, поскольку:

- в обоих протоколах используется свой собственный C-подобный язык для описания передаваемых сообщений, определения которых помещаются в отдельный файл (в XDR – в файл с расширением `.x`, в Protocol Buffers – в файл с расширением `.proto`);
- оба протокола в своей работе используют двоичное кодирование данных.

Стоит отметить, что Protocol Buffers представляет собой более современный протокол передачи данных по сравнению с XDR, непрерывно развивается по сегодняшний день и активно поддерживается со стороны разработчиков. Также имеется возможность его использования в операционной системе Windows. В связи с этим, в качестве достойной альтернативы XDR было принято решение использовать именно Protocol Buffers.

Поскольку библиотека libguestfs реализована на языке C, для замены XDR использовалась реализация Protocol Buffers для языка C – проект protobuf-c[2]. Данный проект не является официально разрабатываемым проектом от Google, но перечислен как один из рекомендуемых[3].

[1] https://developers.google.com/protocol-buffers
[2] https://github.com/protobuf-c/protobuf-c
[3] https://github.com/google/protobuf/wiki/Third-Party-Add-ons

5.1.3 Замена XDR на Protocol Buffers

5.1.3.1 Изменения исходного кода генератора

Процесс сборки проекта libguestfs включает в себя сборку специального генератора. Данный генератор реализован на языке OCaml и предназначен для создания значительной части кода, используемого во время сборки библиотеки libguestfs. Генерируемый код включает в себя код, отвечающий за передачу данных между библиотекой и демоном, заключающуюся в отправке демону специальных команд и возвращению результатов их исполнения.

Все передаваемые между библиотекой и демоном данные представляют собой сериализованные XDR-структуры. Определения этих структур на языке C генерируются с помощью утилиты rpcgen[1] на основе файла формата `.x`, используемого в XDR. Данный файл, в свою очередь, создается в результате вызова генератора.

Для каждой удаленной процедуры, вызываемой со стороны библиотеки для исполнения демоном, генератор хранит информацию, в которой описывается о входных параметрах процедуры и возвращаемом ей значении. На основе этой информации генератор создает файл формата `.x`, в котором записаны определения структур, предназначенных для хранения значений входных аргументов функций, и структур, предназначенных для хранения возвращаемых значений. В рамках решения задачи по замене XDR на Protocol Buffers, в исходный код генератора были добавлена возможность создания файла формата `.proto`, содержащий аналогичные определения структур, но уже написанные на языке, используемом в Protocol Buffers.

Стоит отметить, что структуры формата XDR, предназначенные для передачи входных аргументов функций и результатов их исполнения, используются в дальнейшем в качестве внутренних структур, участвующих в непосредственном выполнении команды. Для того, чтобы не переписывать код, не участвующий в передаче данных, но использующий XDR-структуры, были реализованы специальные конвертеры, переводящие protobuf-структуры в XDR-структуры и обратно.

5.1.3.2 Изменения исходного кода библиотеки libguestfs и демона

Основные функции, отвечающие за взаимодействие демона и библиотеки – отправка команд, возвращение результатов их исполнения, отправка и получение файлов – реализованы в модуле proto.c библиотеки и в модуле proto.c демона. Поскольку работа всех функций основана на использовании XDR-структур, указанные модули были переписаны для возможности использования protobuf -структур.

[1] http://en.wikipedia.org/wiki/RPCGEN

5.2 Передача файлов через общую память

Как упоминалось ранее, «общение» библиотеки с демоном осуществляется посредством сетевого взаимодействия, все данные – команды и результаты их выполнения – передаются через сокеты. В частности, файлы, отправленные демону на запись или полученные в результате выполнения операции чтения, пересылаются частями – максимум по 8 КБ в каждом сообщении. Учитывая, что пересылка каждого сообщения сопровождается дополнительными издержками, связанными с организацией и структурой сетевого взаимодействия (копирование данных для упаковки в пакет, передача пакета, распаковка пакета и извлечение данных), это приводит к существенному замедлению скорости выполнения операций чтения и записи, особенно при передаче файлов большого размера (порядка нескольких десятков мегабайт и выше). С целью повысить скорость работы библиотеки было принято решение организовать передачу файлов через разделяемую между библиотекой и демоном (общую) память.

Использование разделяемой памяти позволит:

- передавать содержимое файлов частями гораздо большего размера без увеличения дополнительных издержек, связанных с передаваемым размером данных;
- избавиться от «лишнего» копирования, происходящего при передаче данных через сокеты.

Схема работы библиотеки при использовании общей памяти представлена на рисунке 4.

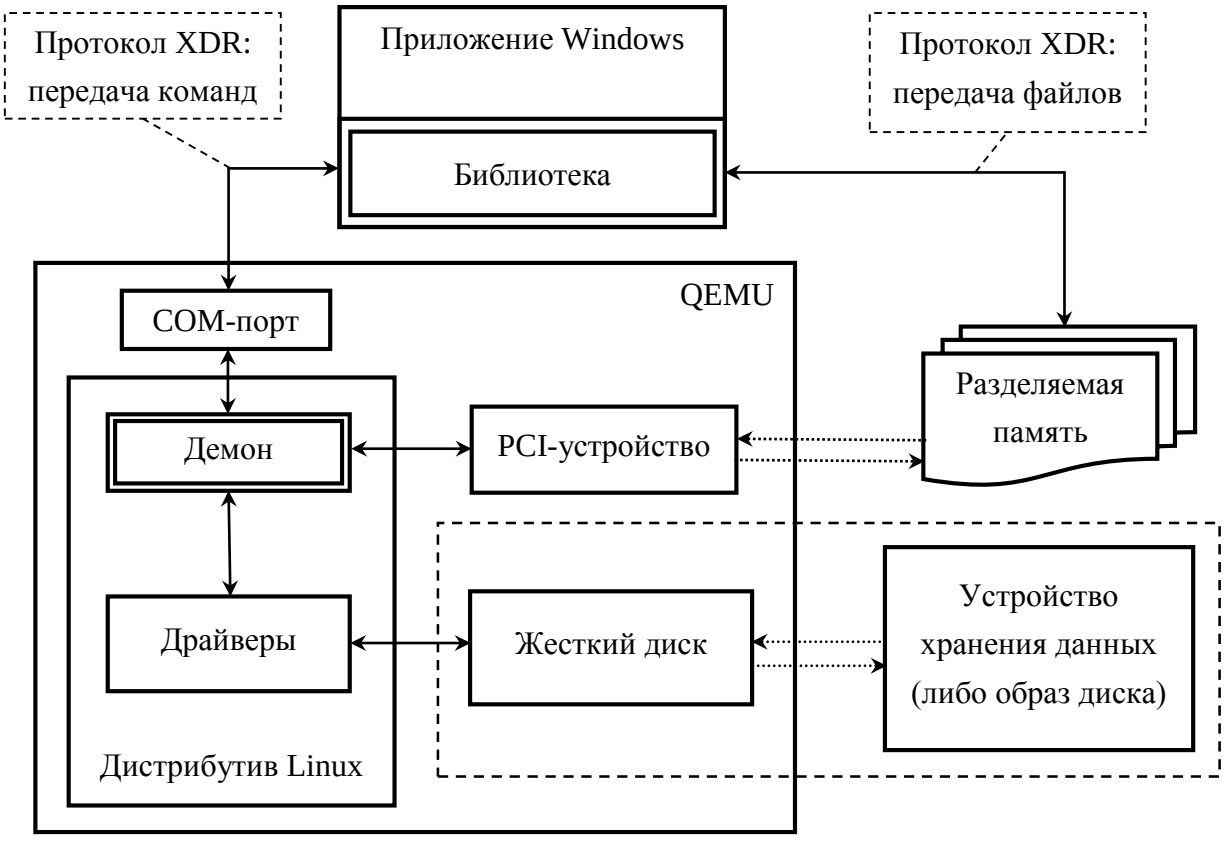


Рисунок 4 – Работа библиотеки при использовании общей памяти

5.2.1 Организация доступа библиотеки к разделяемой памяти

Организация доступа к разделяемой памяти со стороны библиотеки достаточно проста. В операционной системе Windows разделяемая память между процессами организуется с помощью файловых отображений[1]. Файловые отображения относятся к глобальным объектам операционной системы Windows и обладают собственным уникальным именем. Для получения объекта файлового отображения используется вызов функции `OpenFileMapping`, в котором указывается имя файлового отображения. Далее с помощью вызова функции `MapViewOfFile` участок общей памяти отображается в используемое процессом виртуальное адресное пространство, а полученный в результате вызова указатель используется в качестве буфера для передачи и получения файлов.

Для работы с разделяемой памятью на стороне библиотеки была реализована кроссплатформенная структура `os_shared_memory`, которая была добавлена в созданную библиотеку кроссплатформенных интерфейсов. Данная структура позволяет открывать разделяемую память, получать указатель на начало разделяемой памяти, а также хранит сведения о свойствах разделяемой памяти, таких как размер и имя.

```
struct os_shared_memory
{
    const struct os_shared_memory_ops *ops;
};

struct os_shared_memory_ops
{
    int (*open) (struct os_shared_memory *shmem);
    int (*close) (struct os_shared_memory *shmem);
    const char *(*get_name) (struct os_shared_memory *shmem);
    uint64_t (*get_size) (struct os_shared_memory *shmem);
    void * (*get_ptr) (struct os_shared_memory *shmem);
    void (*print) (struct os_shared_memory *shmem, int n);
};
```

В случае операционной системы Windows в качестве структуры `os_shared_memory` выступает структура `windows_shared_memory`, которая предоставляет интерфейс по работе с файловыми отображениями операционной системы Windows.

```
struct windows_shared_memory
{
  const struct os_shared_memory_ops *ops;

  const char *name;
  uint64_t size;

  HANDLE h;
  void *ptr;
};
```

[1] https://msdn.microsoft.com/en-us/library/windows/desktop/aa366878(v=vs.85).aspx

В исходный код библиотеки были внесены следующие изменения:

1. В открытый программный интерфейс библиотеки добавлена функция `guestfs_set_shared_memory`, предоставляющая возможность включить и отключить использование общей памяти для передачи файлов, а также задать имя и размер. Данная функция была «зарегистрирована» в исходном коде генератора в модуле generator/actions.ml:

```
  { defaults with
    name = "set_shared_memory";
    style = RErr, [Bool "enable"], [OInt "size"; OString "name"];
    blocking = false;
    shortdesc = "enable and configure shared memory parameters";
    longdesc = "\
If C<enable> is true, then shared memory is enabled in the
libguestfs appliance. The default is false.

This affects whether files are able transfer across shared memory.

You must call this before calling C<guestfs_launch>, otherwise
it has no effect." };
```

2. Поскольку состояние библиотеки теперь характеризуется использованием общей памяти, в «главный» обработчик `guestfs_h` было добавлено специальное поле `shmem` типа `struct os_shared_memory*` для работы с общей памятью:

```
struct guestfs_h
{
  ...
    /*** Shared memory ***/
  struct os_shared_memory *shmem;
  ...
};
```

3. В процедуру запуска виртуальной машины добавлена проверка необходимости использования разделяемой памяти, и при наличии таковой во входные параметры виртуальной машины включается соответствующая опция:

```
/* Set up ivshmem device */
if (g->shmem) {
  ADD_CMDLINE ("-device");
  ADD_CMDLINE_PRINTF("ivshmem,size=%"PRIu64"M,shm=%s",
                     g->shmem->ops->get_size (g->shmem) >> 20,
                     g->shmem->ops->get_name (g->shmem));
}
```

4. Изменен протокол передачи данных для случая использования общей памяти. Более подробно о изменениях в процедуре передачи данных рассказывается в 5.2.3.

5.2.2 Организация доступа демона к разделяемой памяти

Организация доступа к разделяемой памяти со стороны демона выглядит немного сложнее, чем в случае с библиотекой, работающей на стороне Windows-приложения. Демон работает в гостевой операционной системе Linux внутри виртуальной машины, не имеет доступа к операционной системе Windows, в которой запущена виртуальная машина, и, как следствие, не может напрямую использовать файловые отображения операционной системы Windows. В данном случае необходимо, чтобы запущенная в Windows виртуальная машина QEMU, обладая доступом к разделяемой памяти Windows, представила эту общую память в виде некоторого устройства, доступного демону в операционной системе Linux внутри виртуальной машины.

5.2.2.1 Устройство IVSHMEM виртуальной машины QEMU

В документации к виртуальной машине QEMU [11] указывается о возможности использовать общую память между хостом и гостем. В момент запуска виртуальная машина QEMU запрашивает у операционной системы участок разделяемой памяти определенного размера (согласно указанному при запуске), и внутри виртуальной машины полученная общая память представляется гостевой операционной системе в виде памяти особого PCI-устройства [12], позволяющего приложениям взаимодействовать с ним «без копирования» (zero-copy communication). Для обозначения данного устройства используется аббревиатура IVSHMEM (Inter-VM Shared Memory).

Однако данная возможность доступна только при использовании виртуальной машины QEMU в операционной системе Linux. Поэтому в рамках решения задачи организации доступа демона к разделяемой памяти Windows необходимо реализовать эту возможность при использовании виртуальной машины QEMU в операционной системе Windows.

5.2.2.2 Реализация IVSHMEM-устройства для работы в Windows

В исходном коде проекта QEMU реализация IVSHMEM-устройства содержится в модуле ivshmem.c. Для описания IVSHMEM-устройства и его состояния используется структура `IVShmemState`, наследуемая от структуры `PCIDevice`. Установка разделяемой памяти хоста в качестве памяти IVSHMEM-устройства происходит во время инициализации устройства. Таким образом, для реализации возможности использования IVSHMEM-устройства в Windows необходимо внести изменения в процедуру его инициализации.

Инициализация IVSHMEM-устройства происходит в функции `pci_ivshmem_init`. В исходной реализации функции `pci_ivshmem_init` происходит создание разделяемой POSIX-памяти с помощью вызова функции `shm_open`. Полученный объект разделяемой памяти используется в вызове функции `create_shared_memory_BAR`, внутри которой с помощью функции `mmap` происходит отображение созданной в функции `pci_ivshmem_init` разделяемой памяти в виртуальное адресное пространство QEMU. Полученный в

результате вызова mmap указатель на общую память регистрируется в качестве физической памяти создаваемого PCI-устройства.

Таким образом, основные изменения, внесенные в процесс инициализации IVSHMEM-устройства, связаны со способом получения разделяемой памяти и отображения её в виртуальное адресное пространство QEMU. Во время инициализации IVSHMEM-устройства в функции pci_ivshmem_init происходит создание файлового отображения (разделяемой памяти) операционной системы Windows с помощью функции CreateFileMapping. Имя создаваемого файлового отображения, указываемое при вызове функции CreateFileMapping, передается в качестве входного параметра при запуске виртуальной машины QEMU.

```
static int pci_ivshmem_init(PCIDevice *dev)
{
    IVShmemState *s = IVSHMEM(dev);
    ...
    hMapFile = CreateFileMapping(INVALID_HANDLE_VALUE,
        NULL,
        PAGE_READWRITE,
        0, s->ivshmem_size,
        shmobj_global);
        ...
     create_shared_memory_BAR(s, hMapFile);
    ...
}
```

Полученный в функции pci_ivshmem_init объект файлового отображения используется в функции create_shared_memory_BAR, внутри которой с помощью функции MapViewOfFile возвращается указатель на отображенную в виртуальное адресное пространство QEMU общую память, который регистрируется в качестве физической памяти созданного PCI-устройства.

```
/* create the shared memory BAR when we are not using the server, so we can
 * create the BAR and map the memory immediately */
static void create_shared_memory_BAR(IVShmemState *s, HANDLE hMapFile) {
    LPVOID pBuf;
    ...
    pBuf = MapViewOfFile(hMapFile,
        FILE_MAP_ALL_ACCESS, 0, 0, s->ivshmem_size);
    ...
    memory_region_init_ram_ptr(&s->ivshmem, OBJECT(s), "ivshmem.bar2",
                               s->ivshmem_size, pBuf);
    vmstate_register_ram(&s->ivshmem, DEVICE(s));
    memory_region_add_subregion(&s->bar, 0, &s->ivshmem);

    /* region for shared memory */
    pci_register_bar(PCI_DEVICE(s), 2, s->ivshmem_attr, &s->bar);
}
```

Для реализации IVSHMEM-устройства в Windows использовался исходный код QEMU версии 2.1.2, полученный с официального сайта проекта[1]. Сборка QEMU осуществлялась с использованием MinGW в соответствии с указаниями, представленными на некоторых сайтах проекта[2].

5.2.2.3 Организация доступа демона к IVSHMEM-устройству

Для возможности работы в операционной системе Linux с IVSHMEM-устройством, в используемый в виртуальной машине дистрибутив Linux был добавлен специальный драйвер, который загружается в ядро Linux во время выполнения начального загрузочного скрипта. Данный драйвер реализован в рамках проекта guest_code[3] с использованием фреймворка UIO [13] ядра Linux.

В исходный код демона были внесены следующие изменения:

1. Реализована структура `shared_memory`, предназначенная для работы с памятью IVSHMEM-устройства;
2. В процесс инициализации демона (подготовки демона к работе) добавлена проверка необходимости использования общей памяти, и при наличии таковой осуществляется поиск IVSHMEM-устройства и отображение его памяти в виртуальное адресное пространство демона с помощью функции специальной функции `open`, реализованную в рамках структуры `shared_memory`;
3. Изменен протокол передачи файлов для случая использования общей памяти. Более подробно о изменениях в процедуре передачи данных рассказывается в 5.2.3.

5.2.3 Изменения в протоколе передачи файлов

В исходной реализации файлы между библиотекой и демоном передавались через сокеты по частям, максимум по 8 КБ в каждом сетевом пакете. При таком способе передачи отсутствует необходимость ожидать, пока получателем будет считана и сохранена переданная часть файла, поскольку каждая следующая часть файла передается в отдельном пакете и не перезаписывает предыдущую.

В случае использования общей памяти для передачи файлов это не так. «Передача» очередной части файла осуществляется путем её записи в общую память. Для удобства рассмотрим процедуру передачи файла от библиотеку к демону. Отправитель (в данном случае библиотека) осуществляет запись передаваемой части файла в общую память, после чего отправляет получателю (в данном случае демону) сообщение-уведомление `guestfs_shm_chunk` о том, что «переданная» часть файла готова к получению. После получения уведомления получатель (демон) считывает содержимое общей памяти и

[1] http://wiki.qemu.org/Download
[2] http://wiki.qemu.org/Hosts/W32
http://lassauge.free.fr/qemu/QEMU_on_Windows.html
[3] https://gitorious.org/nahanni/guest-code/

отправляет сообщение-подтверждение об успешном получении переданных данных. Только после получения подтверждения отправитель (библиотека) передает следующую часть файла, поскольку если следующую часть файла будет «отправлена» до момента считывания получателем предыдущей, то новая часть файла перепишет содержимое предыдущей, в результате чего содержимое полученного файла окажется некорректным.

На стороне библиотеки за передачу файлов отвечают глобальные функции `guestfs___send_file` и `guestfs___recv_file`, определенные в модуле proto.c; на стороне демона - функции `send_file_write` и `receive_file`, определенные в модуле daemon/proto.c. Для каждой из указанных функций была добавлена реализация, осуществляющая передачу файлов через разделяемую память с описанным выше механизмом «подтверждения о получении», и вынесена в отдельную функцию. Исходная реализация указанных функций, осуществляющая передачу файлов через сокеты, была также вынесена в отдельную функцию. Поскольку возможность передачи файлов через разделяемую память носит необязательный (опциональный) характер, в зависимости от выбранного режима работы внутри указанных функций осуществляется проверка необходимости использования общей памяти для передачи содержимого файлов, после чего происходит вызов соответствующей реализации функции.

Ниже приведен пример изменения функции `receive_file`, определенной в daemon/proto.c. Если изначально функция `receive_file` имела следующий вид:

```
int
receive_file (receive_cb cb, void *opaque)
{
/* передача файла через сокет */
}
```

то после добавления возможности передачи файлов через общую память реализация функции `receive_file` стала иметь следующий вид:

```
int
receive_file (receive_cb cb, void *opaque)
{
  if (enable_shm)
    /* передача файла через общую память */
    return receive_file_shm (cb, opaque);

  /* передача файла через сокет */
  return receive_file_sock (cb, opaque);
}
```

Аналогичные изменения были выполнены с каждой функцией, «участвующей» в процессе передачи файлов. Описанная выше процедура позволила добавить возможность передачи файлов через разделяемую память, сохранив используемый внутри библиотеки (и демоне) программный интерфейс.

6 Тестирование и анализ результатов

6.1 Цели тестирования

Главными целями тестирования является проверка работоспособности полученной библиотеки и оценка изменения производительности работы библиотеки libguestfs, портированной для работы в операционной системе Windows, по сравнению с исходной, предназначенной для работы в Linux. Производительность работы библиотеки оценивается временем выполнения следующих операций:

- запуск виртуальной машины и полная инициализация «устройства»;
- монтирование образа диска с требуемой файловой системой;
- чтение файловой системы;
- запись файловой системы.

Также необходимо проверить как предложенные в настоящей работе в 5.1 и 5.2 оптимизации влияют на скорость выполнения операций чтения и записи файловой системы.

6.2 Сценарии тестирования

Тестирование производительности работы полученной библиотеки проводилось в несколько этапов.

На первом этапе было выполнено сравнение производительности работы полученной портированной библиотеки libguestfs в операционной системе Windows и исходной библиотеки libguestfs в операционной системе Linux. Тестирование производительности включало в себя:

1. Запуск виртуальной машины с 512 МБ памяти;
2. Монтирование образа диска размера 4 ГБ с файловой системой Ext2;
3. Чтение 1 файла размером 1.5 ГБ;
4. Чтение 10000 файлов размером 4 КБ каждый;
5. Запись 1 файла размером 1.5 ГБ;
6. Запись 10000 файлов размером 4 КБ каждый.

На втором этапе было выполнено сравнение скорости выполнения операций чтения и записи при использовании протокола XDR и при использовании протокола Protocol Buffers как в операционной системе Windows, так и в операционной системе Linux. Тестирование скорости выполнения операций чтения и записи включало в себя:

1. Чтение 1 файла размером 1.5 ГБ;
2. Чтение 10000 файлов размером 4 КБ каждый;
3. Запись 1 файла размером 1.5 ГБ;
4. Запись 10000 файлов размером 4 КБ каждый.

На третьем этапе было выполнено сравнение скорости выполнения операций чтения и записи при использовании для передачи файлов разделяемой памяти и без неё как в операционной системе Windows, так и в операционной системе Linux. Тестирование скорости выполнения операций чтения и записи включало в себя:

1. Чтение 1 файла размером 1.5 ГБ;
2. Чтение 10000 файлов размером 4 КБ каждый;
3. Запись 1 файла размером 1.5 ГБ;
4. Запись 10000 файлов размером 4 КБ каждый.

Измерения проводились на компьютере со следующими характеристиками:

- процессор Intel Core i7 с частотой 2.5 ГГц;
- оперативная память 16 Гб.

6.3 Результаты тестирования

Ниже в таблицах 2,3 и 4 приведены результаты тестирования.

6.3.1 Производительность работы библиотеки в Linux и Windows

Таблица 2 – Результаты тестирования

	Linux	Windows
Запуск виртуальной машины	8.1 сек.	15.8 сек.
Монтирование файловой системы	7.9 сек.	11.3 сек.
Чтение 1 ф. х 1.5 ГБ	46.1 сек.	47.8 сек.
Чтение 10000 ф. х 4 КБ	16.4 сек.	17.9 сек.
Запись 1 ф. х 1.5 ГБ	51.0 сек.	51.0 сек.
Запись 10000 ф. х 4 КБ	24.3 сек.	26.4 сек.

6.3.2 Скорость выполнения операций чтения и записи при использовании XDR и при использовании Protocol Buffers

Таблица 3 – Результаты тестирования

	Linux: XDR	Linux: Protocol Buffers	Windows: XDR	Windows: Protocol Buffers
Чтение 1 ф. х 1.5 ГБ	46.1 сек.	45.5 сек.	47.8 сек.	48.2 сек.
Чтение 10000 ф. х 4 КБ	16.4 сек.	16.7 сек.	17.9 сек.	17.5 сек.
Запись 1 ф. х 1.5 ГБ	51.0 сек.	51.3 сек.	51.0 сек.	52.0 сек.
Запись 10000 ф. х 4 КБ	24.3 сек.	24.2 сек.	26.4 сек.	25.8 сек.

6.3.3 Скорость выполнения операций чтения и записи при использовании общей памяти и без использования общей памяти

Таблица 4 – Результаты тестирования

	Linux	Linux: общая память	Windows	Windows: общая память
Чтение 1 ф. х 1.5 ГБ	46.1 сек.	13.3 сек.	47.8 сек.	13.0 сек.
Чтение 10000 ф. х 4 КБ	16.4 сек.	16.1 сек.	17.9 сек.	17.3 сек.
Запись 1 ф. х 1.5 ГБ	51.0 сек.	25.7 сек.	51.0 сек.	27.3 сек.
Запись 10000 ф. х 4 КБ	24.3 сек.	24.2 сек.	26.4 сек.	27.2 сек.

6.4 Анализ результатов

На основе полученных результатов можно сделать следующие выводы:

1. Производительность работы портированной в Windows версии библиотеки libguestfs немного хуже исходной. Данное падение производительности связано, главным образом, с особенностями работы виртуальной машины QEMU в операционных системах Windows и Linux, о чем свидетельствует возросшее время запуска и монтирования файловой системы. Скорость выполнения операций чтения и записи снизилась незначительно, что связано с особенностями реализации сокетов и TCP протокола в операционных системах Windows и Linux;
2. Замена протокола сериализации данных с XDR на Protocol Buffers не оказывает влияния на скорость выполнения операций чтения и записи и, таким образом, не приносит выигрыша в производительности;
3. Использование разделяемой памяти для передачи файлов позволяет увеличить скорость выполнения операций чтения и записи, особенности в случае файлов большого размера. В случае передачи файлов малого размера (до 8 КБ), содержимое которых помещается в один пакет (как при передаче без использования общей памяти), скорость выполнения операций чтения/записи существенно не изменяется. В случае передачи файлов большого размера (порядка нескольких десятков мегабайт и выше), происходит значительный прирост в производительности: скорость записи возрастает почти в 2 раза, скорость чтения – более, чем в 3 раза. Стоит отметить, что в случае использования общей памяти скорость передачи файлов также существенно зависит от размера общей памяти – чем больше размер используемой разделяемой памяти, тем выше скорость передачи файлов. Причем прирост в скорости передачи файлов от увеличения размера используемой общей памяти будет происходить только тогда, пока размер общей памяти не превосходит размер передаваемого файла.

7 Результаты работы

В настоящей работе получена библиотека, позволяющая приложениям операционной системы Windows работать с нативными файловыми системами операционной системы Linux, которые не поддерживаются операционной системой Windows. Полученная библиотека представляет собой портированную в Windows версию библиотеки libguestfs, реализованную в рамках одноименного проекта компании RedHat.

Организация доступа к файловым системам основана на использовании нативных драйверов операционной системы Linux, которые включены в её основное ядро. Для возможности работы этих драйверов в операционной системе Windows окружение Linux эмулируется с помощью виртуальной машины. В качестве виртуальной машины используется QEMU.

Для повышения скорости выполнения операций чтения и записи реализована возможность передачи файлов через общую память.

8 Библиография

[1] Wikipedia [Электронный ресурс]. – Режим доступа : https://www.wikipedia.org, свободный. – Загл. с экрана.

[2] R. Srinivasan. XDR: External Data Representation Standard. – RFC 1832. – Sun Microsystems, 1995.

[3] Werner Almesberg. Booting Linux: The History and The Future. – Proceedings of the Ottawa Linux Symposium, 2000.

[4] Libguestfs : Documentation [Электронный ресурс]. – Режим доступа : http://libguestfs.org/guestfs.3.html, свободный. – Загл. с экрана.

[5] Linux man pages [Электронный ресурс]. – Режим доступа : http://linux.die.net/man, свободный. – Загл. с экрана.

[6] Gary V. Vaughan, Ben Elliston, Tom Tromey, Ian Lance Taylor. GNU Autoconf, Automake, and Libtool. – New Riders, 2000

[7] David MacKenzie, Ben Eliston,Akim Demaille, GNU Autoconf. – Free Software Foundation, Inc., 2012

[8] David MacKenzie, Tom Tromey, Alexandre Duret-Lutz, Ralf Wildenhues, Stefano Lattarini. GNU Automake. – Free Software Foundation, Inc., 2014

[9] Gordon Matzigkeit, Alexandre Oliva, Thomas Tanner, Gary V. Vaughan. GNU Libtool. – Free Software Foundation, Inc., 2015

[10] M. Eisler. XDR: External Data Representation Standard. – RFC 4506. – Network Appliance, Inc., 2006.

[11] QEMU Emulator User Documentation [Электронный ресурс]. – Режим доступа : http://qemu.weilnetz.de/qemu-doc.html, свободный. – Загл. с экрана.

[12] Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman. Linux Device Drivers, Third Edition. – O'Reilly Media, Inc., 2005.

[13] H. Koch. Userspace I/O drivers in a realtime context, in Proceedings of the 11th Real-Time Linux Workshop. – Dresden, Germany : OSADL, 2009